UiO : Department of Physics
University of Oslo

On-Board Controller & Data Handling

A CubeSTAR Subsystem

Master Thesis

Joakim Myrland

February 2013

Abstract

This thesis discusses the integration of the subsystems in a CubeSat satellite and the implementation of a central “On-Board Controller & Data Handling” system. Potential radiation problems are investigated and addressed.

CubeSTAR is a CubeSat satellite developed at the University of Oslo. The satellite requires some method of handling commands and sending data to the ground station. The satellite needs to be autonomous, so a method of coordinating the operation of each subsystem is required.

The On-Board Controller integrates all the satellite’s subsystems to a unified control- and monitoring system. It will make sure each subsystem performs its operation and attempts recovery if something is out of the ordinary.

Space radiation can cause memory bit-flips and other effects on electronics within a satellite. In case a subsystem does not function as expected or has disruptive behavior, it will need to be recovered or disabled.

The On-Board Controller addresses the memory bit-flip problem caused by space radiation by using a new memory technology (MRAM) and a special 8-bit hamming code for commands and variables.

Preface

This thesis is the conclusion of a Master of Science degree in Electronics and Computer Technology at University of Oslo. The work on this thesis was completed in the period between February 2010 and February 2012, where the first year was part time and the second year was full time.

The reader of this thesis should be aware of the following:

- Acronyms will only be defined the first time they are mentioned in the thesis. A list of acronyms can be found in the Index at the end of the last chapter
- Several acronyms are considered "general knowledge" and are only defined in the index. E.g., CMOS and I^2C.
- It is assumed the reader understands basic technical terms like pF, byte, oscillator and similar.
- Application Notes and Datasheets are not included in the list of references.

Acknowledgements

I would like to thank my advisor Torfinn Lindem for introducing me to this exciting project and his help and guidance throughout the work. Appreciations goes out to Stein Lyng Nielsen at the Electronics Laboratory for his help in PCB design and layout. I would like to thank Johan Tresvig and Knut Olav Skyttemyr for the collaboration, suggestions and ideas on the project. Last but not least, big thanks to everyone who has helped review this thesis.

Oslo, Norway, February 2012
Joakim Myrland

Contents

Chapter 1:

Introduction

1.1. STAR INITIATIVE

Space Technology and Research (STAR) is a strategic initiative by the Faculty of Mathematics and Natural Sciences of the University of Oslo (UiO). The main aim of STAR is to achieve full-scale integrated collaboration between space plasma experimentalists, theoreticians and numerical specialists in order to implement the next generation of space instruments.

STAR's focus is on plasma research, with particular emphasis on problems relevant to the satellite and rocket activities within STAR. The experimental part of STAR is focused on cusp, polar cap and magnetospheric boundary layer phenomena studied by in-situ and remote sensing techniques.

The group play the leading role in the ICI (Investigation of Cusp Irregularities) sounding rocket program [1], to explore instability phenomena in the auroral cusp causing high-frequency backscatter irregularity formation and GPS scintillations etc. This activity is inclined to contribute to ESA's Space Situation Awareness program, which is expected to become a rapid growing activity within Europe.

1.2. CUBESTAR PROJECT

CubeSTAR is a small satellite based on the CubeSat standard [2]. This standard makes it possible to develop low cost satellites, with "piggyback" opportunities for commercial space launches. CubeSTAR will use a multi-Needle Langmuir Probe system (m-NLP) [3], similar to the ones used by the ICI sounding rocket program. The mission objective for CubeSTAR is to gather scientific data from the ionospheres F-layer for a minimum of 2 months.

CubeSTAR consists of several subsystems that provide the functionality and operation of the satellite. These subsystems include:

- A Communication system with antenna (COMM) [4].
- An Electrical Power System (EPS).
- An On-Board Controller & Data Handling system commonly referred to as the On-Board Controller (OBC).
- An Attitude Determination & Control System (ADCS).
- The scientific payload using m-NLP.

1.2.1. Scientific Background

The ionosphere is a region of the atmosphere that is filled with plasma. This plasma is created by solar radiation, in which the density varies according to the amount of plasma received from the Sun. The plasma layers are affected by both day and night cycles, as well as yearly seasons and solar storms.[1]

Solar storms will create heavy turbulence in the plasma that cause scintillation and degradation in satellite signals. By taking measurements of the plasma density, researchers should gain more insight into this phenomenon and be able to provide useful information to improve the reliability of satellite communication.

1.2.2. What is new?

Most electron measurements done in the ionosphere today are called Total Electron Content (TEC) and is a term for electrons/m^2. They are a major factor in current space weather models. You can find TEC by taking the line integral of the plasma density (electrons/m^3). The most common method of measuring TEC is by taking the difference in range (phase delays) of the two frequencies sent by GPS satellites [5].

CubeSTAR will measure the electron density in-situ using a new technique with m-NLP. This will show structures in the electron clouds, which you cannot see with TEC. The m-NLP will improve the spatial resolution compared to single Langmuir probe measurements, from today's seven kilometers down to the meter level.

[1] Even though the plasma is mostly affected by solar activity, there are some other known factors, including disruptions before and during earthquakes and tsunamis. [43]

1.3. GOALS OF THE THESIS

This thesis puts an emphasis on the integration of the subsystems in the CubeSTAR satellite. This includes:

- The creation of an internal communication protocol, with all commands to and from each subsystem.
- The development of software that handles the commands and control of each subsystem.
- The creation of hardware that can run the software and be able to tolerate the satellite's radiation environment.
- The creation of integration tools to test and verify internal communication.
- Identifying radiation effects and implementing methods in software to detect, report and mitigate these.

For the satellite to meet operational requirements, the ground crew will need to receive scientific data. The ground crew should be continually updated on what is happening on the satellite and if everything is running as normal. Creating a system to handle these tasks is a fundamental part of the satellite's operation.

When a satellite is deployed and is powered up for the first time, there is a strict guideline for startup procedures. This includes regulations on when deployment of antenna and scientific probes can occur, as well as when radio signals can be sent from the satellite. The launch provider requires a method of supervising these operations.

As it will not be possible to repair malfunctioning subsystems in space, the system should be as reliable as possible. Design errors, space radiation and other factors can cause unexpected behavior in the satellite. A system that can attempt recovery, or maintain limited operation should be implemented. If this is not in place once such an error occurs, the satellite will most likely fail to meet the operational requirements.

Autonomous operation of the satellite is important, as the satellite will be out of reach from the ground station during most of its orbit cycle. Most tasks are time critical and cannot wait for a user on the ground to make a decision.

The software for the controller will need to address the following problems:

- Handle all command and data transfers to, from and between all subsystems.
- Monitor and maintain the general operation and health of the satellite, which includes startup and shutdown procedures.
- Send a periodic beacon signal and keep a log of events and historical data until a downlink of the data is completed.
- Provide an accurate time reference for all events and scientific data.

Chapter 2:

Radiation Effects in Space

Space radiation is used as a term for energetic particles, often traveling at high speeds through space. These particles come in the form of nuclei/ions, electrons, protons or neutrons. Their origin is primarily from the Sun, but also from deep space [6].

There is a significant amount of radiation in space compared to the surface of the Earth. Radiation is very hostile to electronics and will eventually cause components to malfunction. There is no method of repair. This means the electronics needs to be radiation tolerant so it can remain operational during the satellite's mission. Implementing redundancy will also help ensure continual operation.

The main categories of space radiation in Low Earth Orbit (LEO) are solar flare particles, cosmic rays and trapped radiation belt particles [7]. Solar flares are bursts of particles from the Sun, which causes an increase in the normal amount of radiation around Earth. Cosmic Rays are the general background radiation and can even be observed on the surface of the Earth. It exists throughout the universe. Trapped radiation belt particles are fields of plasma created when solar flare particles and cosmic rays enter a planet's magnetic field. On Earth, this is known as the inner and outer Van Allen belts and is a major factor when considering radiation effects on satellites in Earth orbit.

A particle needs to have enough energy to penetrate a satellite's shielding and still contain energy higher than the 'threshold' of the electronic circuit to cause any effects. Because of this, most satellites have significant amount of shielding around critical components and systems. This will limit the amount

and reduce the energy of particles hitting the electronic components within it.

2.1. SINGLE EVENT UPSET

A Single Event Upset (SEU) is a non-destructive error caused by a high-energy particle interacting with an electronic device [8]. The most common effect of this is a bit-flip in a memory cell or a register. These errors can be corrected with memory scrubbing (a rewrite of the bit) or restart of the device.

As this could cause undesired behavior in the affected electronics and corrupt data, methods to circumvent these effects have been developed. These are covered in more detail in “Chapter 3: Redundancy Schemes”.

In LEO, trapped radiation belt protons are the most important source for SEUs [9]. Another important factor are solar flares that occur during a Solar Proton Event (SPE). During a SPE, the flux is typically increased by a factor from ten to thirty, or with unusually large flares up to several thousands. Cosmic rays have increased significance at higher altitudes [10].

SEUs in LEO are usually observed in the South Atlantic Anomaly (SAA) [11], but during SPEs they are normal in both the north and south poles. The SEU rate scales with particle flux and since flux increases with altitudes in LEO, so do SEUs.

2.1.1. Calculating the Upset Rate

The SEU sensitivity of a device is described by its cross section. A SEU cross section is the number of SEUs (N_{SEU}) divided by fluence (ϕ), where the particles have a given MeV:

$$\sigma = \frac{N_{SEU}}{\Phi}$$

The cross section can be found using a particle accelerator. It is common to plot several cross sections as a function of particle MeV. By using advanced radiation models (SPENVIS), you can find the particle flux as a function of MeV for a given orbit. Using the cross sections and the flux levels for the orbit, you can estimate the error rate in a device as a function of time.

2.2. SINGLE EVENT LATCHUP

A Single Event Latch-up (SEL) is a loss of device functionality, caused by a high-energy particle, resulting in an abnormal high-current state. This could occur when a particle induces the creation of a “short circuit” to ground. These errors can be cleared by power toggling the device, but if this does not happen fast enough the device may suffer permanent damage.

Heavy ion cosmic rays are the primary source for SEL. Cosmic rays become scarcer at low altitudes and are almost non-existent in the ionosphere. The only exception to this is the SAA. The method of finding the SEL cross section follows the same principles of the SEU cross section, but is a function of particle LET.

2.2.1. Other Single Event Effects

Single Event Effects (SEE) such as Single Event Burnout (SEB) and Single Event Gate Rupture (SEGR) describe when high current states or conducting paths are created in a power transistor or power MOSFET [12]. These errors will cause permanent damage.

2.3. TOTAL IONIZING DOSE

Total Ionizing Dose (TID) is the long-term effect of radiation and ultimately decides the lifetime of electronics in space. It is possible to extend the lifetime of a satellite by using shielding and electronic components with high TID rating, but there is no method of avoiding this effect completely.

The primary source for TID comes from protons and electrons. In LEO most of the effect is accumulated during SPEs and in the SAA.

TID for LEO below 500 km, with Inclination (20 < I > 85 degrees) is expected to be between 1k and 10k rad (Si) per year [13]. Commercial ICs typically fail after 3-30 krad of radiation [14] [15]. For a satellite with orbit below 500 km altitude, the commercial ICs should not experience failure from TID for at least 3 to 4 months.

2.4. EFFECTS ON IC TECHNOLOGIES

The effects of radiation depends significantly on the silicon technology and manufacturing process. Different types of technologies will have different types of radiation characteristics, both when it comes to vulnerabilities and resistances.

In CMOS, the technological trend has been a steady decrease in cell size. When looking at radiation effects, you can find two noticeable changes in radiation tolerance. As CMOS manufacturing processes become smaller, the newer CMOS designs tend to have an increased tolerance for TID and a decreased tolerance for SEE [16].

Comparing two CMOS technologies, DRAM and SRAM, we can see a difference in radiation tolerance trends. The Soft Error Rate (SER) trend for SRAM is that the SER has increased as cell size has decreased, while in DRAM the SER has decreased as cell size has decreased. This shows that other factors such as how the memory cells are designed have an impact on radiation tolerance as well. In this instance, the DRAM characteristics are improved, because the Sensitive Volume (SV) is decreased, while the capacitance remains the same.

EEPROM and Flash memories use FGMOS cells and the behavior for these types of memories are therefore different from SRAM or DRAM. Essentially the device type, technology and injected charge will define if and what type of SEE is triggered.

In Commercial Off-The-Shelf (COTS) components, radiation response will vary on factors such as substrate types, dopant levels, processing temperatures, packaging and so on. Variation between foundries and even individual chips on the same silicon wafer can be seen in commercial IC production. This means you cannot find any deterministic radiation behavior with commercial components.

There are some methods to mitigate SEEs in electronic circuits. These methods are called radiation hardening, where the manufacturing technique, silicon design and circuit architecture is changed to limit or remove the radiation effects. COTS companies do not create their electronics with this in mind, as these methods often take up more space and sometimes have lower performance.

Chapter 3:

Redundancy Schemes

Due to the effects of radiation, any electronics exposed to it should implement some methods to limit and avoid the effects. As the components in a satellite can have a wide variety of radiation sensitive devices, the radiation environment needs to be examined. Before this is completed, it is difficult to know what redundancy schemes should be implemented. What are the potential errors? How likely are they to happen?

3.1. HARDWARE

Redundancy in hardware is often implemented when the electronic components are not guaranteed to be reliable or accurate enough for its operating environment.

3.1.1. Triple Modular Redundancy

Triple modular redundancy uses three identical modules that does the same task and compares the different outputs. In order to get an approved output, two of the outputs needs to be equal. This is done by a voting system, but the voting unit is usually a vulnerability point.

3.1.2. Comparison

By using comparison, two results can be compared to each other and need to be equal for the result to be accepted.

3.1.3. Standby Spare

A standby spare is a complete hardware replacement often used for critical parts or systems in a satellite. When a unit malfunctions without possibility of recovery, it will be powered off. The standby spare is then powered on to take its place.

3.1.4. Watchdog

A watchdog timer is a backup functionality intended to avoid system lockup and restart a system that stops running. The watchdog timer is a time delayed restart event; once the timer reaches a certain level, the watchdog performs a restart operation of the system it monitors. In order to avoid this, the watchdog timer counter needs to be reset at regular intervals, so it never reaches the threshold level.

3.2. SOFTWARE

Error Detection and Correction (EDAC) are coding techniques that can detect and correct errors in digital data. Error Detection allows the receiver to know if the data is corrupt, while Error Correction makes it possible to reconstruct the original data [17].

3.2.1. Parity

This is the smallest type of error detection and detects every other bit error. It adds an additional bit to the data and the sum of all ones including the parity bit must be an even number. This is often used when sending individual data bytes on a potentially noisy medium. Since this provides no error correction, it is only useful when there is no requirement that the data is received successfully.

3.2.2. Cyclic Redundancy Checking

Cyclic Redundancy Checking (CRC) uses algorithms to generate a small unique value from a large bulk of data. Any change in the data will alter the CRC value. This method does not allow for error correction, only detection, but the overhead is very small compared to the informational data. It does not alter the original data; it will only add additional bytes at the end of the data segment.

3.2.3. Hamming Code

Hamming code was the first EDAC technique used in computers. It was developed to counteract the read errors experienced in card readers while running old punch card programs. Hamming code achieves the highest possible ratio between informational bits and check bits where the minimum length and distance is three. This means hamming codes are as efficient as possible under those criteria.

The mathematics behind hamming codes are simple. We can find a set of hamming values using these simple rules [18]:

If you want your hamming code to contain X data bits, rewrite the value 'X' into an exponent of two (i.e. $X=2^Y$). Take the exponent 'Y' and add 1 to find the number of check bits required. In other words, the number of check bits is equal to Y+1. This leaves you with a hamming code that is 2^Y+Y+1 bits wide, where the number of data bits is 2^Y.

Using the formula 2^Y+Y+1, we can find the hamming code width using several values for Y. This formula is valid only when $Y \geq 2$.

If Y is three the hamming code will have a bit width equal to $2^3+3+1 = 12$ and contain eight data bits. The smallest hamming code that can attain the hamming bound is found when Y equals two. This gives a hamming code with $2^2+2+1 = 7$ bits and it will contain $2^2 = 4$ data bits.

All check bits have a specific position and are found at the bit positions with value 2^Y. Y can be any number starting at zero. Using the hamming code with four data bits, we get check bit C_0 at position 1, C_1 at position 2, C_2 at position 4 and data bits D_0 at position 3, D_1 at position 5, D_2 at position 6 and D_3 at position 7 (where least significant bit is position 1).

The check bits are special parity bits that cover different parts of the data. The check bits will cover the bits starting at its own position and for bit C_0 include every other bit, C_1 every other 2 bits, C_2 every other 4 bits and so on until all check bits are covered. The check bits themselves are included in this parity check and have a starting value of zero. You can find what bits are covered by each check bit in "Table 3-1 - Check Bit Coverage".

Table 3-1 - Check Bit Coverage

Position	7	6	5	4	3	2	1
Digit	6	5	4	3	2	1	0
Bits	D_3	D_2	D_1	C_2	D_0	C_1	C_0
C_0	X		X		X		X
C_1	X	X			X	X	
C_2	X	X	X	X			

Since we know that all check bits start at zero, they will not affect the parity value anyway and we can remove the check bits from the table. This leaves us with a simpler version as shown in “Table 3-2 - Check Bit Coverage, Simplified”.

Table 3-2 - Check Bit Coverage, Simplified

	D_3	D_2	D_1	D_0
C_0	X		X	X
C_1	X	X		X
C_2	X	X	X	

As an example, a data value of 0001 means that the parity for C_0 is $D_0+D_1+D_3$. We get 1+0+0 = 1. This gives an odd number, which means C_0 is 1. Putting the rest of the check bits into “Table 3-3 - Calculating Hamming Value”, we find that data value of 0001 gives hamming value of 000 0111.

Table 3-3 - Calculating Hamming Value

Digit	7	6	5	4	3	2	1
Code	0	0	0	0	1	1	1
Bits	D_3	D_2	D_1	C_2	D_0	C_1	C_0
				$D_3+D_2+D_1$ = 0+0+0		$D_3+D_2+D_0$ = 0+0+1	$D_3+D_1+D_0$ = 0+0+1

3.2.4. Reed-Solomon Code

Reed-Solomon and BCH codes are often used in space applications due to burst and multi bit correction within larger packets of data. It only works with data packages that have a predetermined length, but within a communication protocol, this is not necessarily an issue. It is widely used in consumer electronics such as DVDs and hard disk drives [19].

3.2.5. Memory Scrubbing

Memory scrubbing is a method where the memory is periodically compared against a checksum and reloaded if an error is found. By checking the memory frequently, it allows the error-correcting code to recover a faulty value before additional bit errors have the chance to occur. A common error-correcting code for memory scrubbing is hamming code and CRC.

Chapter 4:

Operational Requirements

4.1. CUBESTAR REQUIREMENTS

CubeSTAR is a 2-Unit CubeSat structure. This means the satellite is a cube of 10x10x20 cm, with a maximum weight of 2 kg [2] [20]. CubeSTAR will be deployed from a P-POD, which is the standard CubeSat deployment system.

When designing a CubeSat, there are several requirements in the standard to consider. The following list shows some of the requirements:

- No electronics shall be active during launch. All systems shall be turned off, including real time clocks. Batteries shall be fully deactivated during launch or launch with discharged batteries.
- CubeSats with batteries shall have the capability to receive a transmitter shutdown command.
- All deployable such as booms, antennas and solar panels shall wait to deploy a minimum of 30 minutes after the P-POD ejection.
- RF transmitters greater than 1 mW shall wait to transmit a minimum of 30 minutes after the P-POD ejection.

The primary objective for the satellite is to gather scientific data using a m-NLP instrument and transmit this data to the ground station. This data should be sent upon request from the ground station. In order to facilitate up- and downlink operations, the satellite needs a communication system.

A mix of several factors, including the variations in the Earth's gravitational field, will cause a satellite to drift out of position. When the satellite travels through the electron clouds, it will create some local turbulence and disrupt

the electron density around the spacecraft. In order to get proper scientific measurements, the m-NLP needs to be placed in the front of the spacecraft, facing in the direction the satellite is traveling. This means the satellite requires an ADCS that can measure the position and rotate the satellite to the desired orientation.

Another important function is to provide a stable power source for the electronics. This requires solar panels and batteries for continuous operation. A method of monitoring and controlling the distribution of power is required.

Finally, these systems needs to coordinate and perform their operations in proper sequence. Particularly in regard to up and downlink operations, payload dependencies (e.g. an accurate time source) and the operational requirements.

4.2. CUBESTAR SUBSYSTEMS

CubeSTAR's architecture is designed to take advantage of having distributed subsystems, while using a central controller. This is made possible by using a custom CubeSTAR I2C bus protocol. The implementation of that protocol will be explained in "6.4 I^2C BUS COMMUNICATION".

Having distributed subsystems means the satellite's subsystems will perform their tasks autonomously and does not require control or management of its functions from an external source. By having a central controller, the systems will be monitored, information will be collected, system parameters can be altered and commands can be sent in specific situations.

On the CubeSTAR satellite, there are five subsystems:

An ADCS that will handle the satellite's orientation. This is important in order to have the proper direction for the antenna and payload instruments. Once the ADCS is powered on, it will operate entirely without intervention from any other subsystem. Commands can be given to the ADCS and it is expected to act and respond to these inputs. It is however not required for it to perform its primary function, which is to keep the satellite's orientation within accepted limits.

The COMM system will handle the link between the satellite and external sources. All communication to and from the satellite will go through this subsystem. As long as the COMM system is powered on, it will be able to

receive external commands and forward them to any internal subsystem. It will be able to collect and transmit data back to the ground station upon request. In normal operating mode, it will only transmit data when it receives a package from the OBC.

The EPS manages the solar panels and batteries and monitors the power consumption in order to keep the critical systems operational at all times. The EPS enables the on and off power switches for all the subsystems, but will only shut off subsystems on its own in critical situations. During normal operation, the EPS is a passive subsystem that will collect sensor data.

The OBC monitors, collects data and sends commands to the other subsystems. Its functions include:

- Compiling and sending a beacon package to the COMM system.
- Powering on all subsystems through the EPS.
- Check that all subsystems are able to perform their tasks.
- Collect debug information from subsystems.
- Restart or shut off subsystems that are malfunctioning.
- Forward payload and housekeeping data to the COMM system, on request from the ground station.

The m-NLP payload will collect scientific data and store it until a subsystem requests the data. It will need to be powered on before it can begin its operation, but requires no further interaction to perform its function.

4.3. ON-BOARD CONTROLLER OBJECTIVES

The purpose of the OBC is to provide functions that integrate the subsystems in the satellite to a unified control and monitoring system. These functions require knowledge of events that could not be known by a single subsystem, such as tasks that needs to be performed in proper sequence.

When the satellite is ejected from the P-POD, the OBC will need to handle the proper startup sequence. The sequence of when subsystems are powered on is important (e.g., the payload has no function until the satellite has de-tumbled and the attitude vector is within the specified range). According to specifications in the CubeSat standard, the satellite cannot deploy probes or antennas until 30 minutes has passed. Since the OBC is the only subsystem

with non-volatile data memory that could know when the ejection occurred, this is a required function.

The second function of the OBC is to gather sensor data and monitor satellite health. The OBC will continually gather and store housekeeping data in the memory and log all events that occurs in the satellite. This is important in order for the ground crew to have access to detailed information of what has occurred in the satellite.

The satellite may experiences unexpected behavior or errors within subsystems. The OBC is the only source for detecting errors and the only method for recovery. If a subsystem stops responding, or is not working properly, the OBC will need to attempt a recovery or disable the subsystem. Since each subsystem can perform its operation without any intervention from the OBC, the satellite will remain autonomous as long as each subsystem is able to perform its tasks.

The scientific payload has some functional requirements to perform its operation. The ADCS needs to keep the satellite's orientation within a few degrees accuracy. Before any measurements can be done, the m-NLP needs to be deployed. Finally, the payload needs to get an accurate time reference from the OBC for its measurements. Until all of these criteria are met, the payload will not need to be operational and the OBC can keep the payload powered off.

Finally, the OBC will handle all operations between the satellite and the ground station. This is primarily the routine collection of scientific data from the payload and transmitting it to the ground. However, in some cases the historical diagnostic data is desired, as well as the event history. Other functions such as automatic clock synchronization and direct ground crew commands are handled by the OBC.

4.4. RADIATION EFFECTS IN LOW EARTH ORBIT

The CubeSTAR mission orbit will be in the 300km to 500km altitude range. This leaves solar flare particles and cosmic ray effects minimal as they either are deflected by the Earth's magnetic field, or are trapped in the Van Allen belt. Due to the high inclination, the satellite will however travel through the SAA regularly.

The SAA is an area in the South Atlantic Ocean with weak geomagnetic field. This region is where the inner Van Allen belt reaches its lowest altitude at about 200km. It will contain high concentrations of trapped radiation belt particles and allows solar flare particles and cosmic rays to reach lower into the atmosphere.

Since CubeSTAR does not have specific shielding, it is for simplicity assumed the structure, solar panels and PCB's create an equivalent of 1 mm aluminum shielding. A proton with energy of 10 MeV will lose about 9 MeV energy when passing through 1 mm of aluminum [21]. Based on this there's no reason to look for SEEs caused by electrons, or any protons with energy below 10 MeV. With SPENVIS we can find the max proton flux in the SAA for different altitudes [22]. SPENVIS is an advanced space-environment analysis tool developed by ESA.

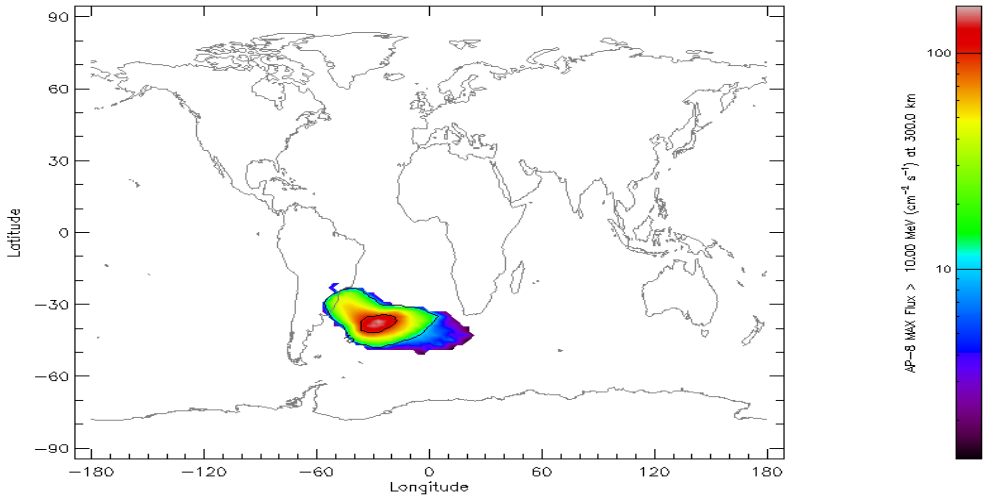


Figure 4-1 - 300km Orbit, >10MeV Proton Flux

"Figure 4-1" and "Figure 4-2" shows that the expected flux rate can reach up to 200-1000 particles/cm^2-second in the SAA, depending on orbit. Because a full radiation analysis is far beyond the scope of this thesis, an example with test results from a different satellite will be shown.

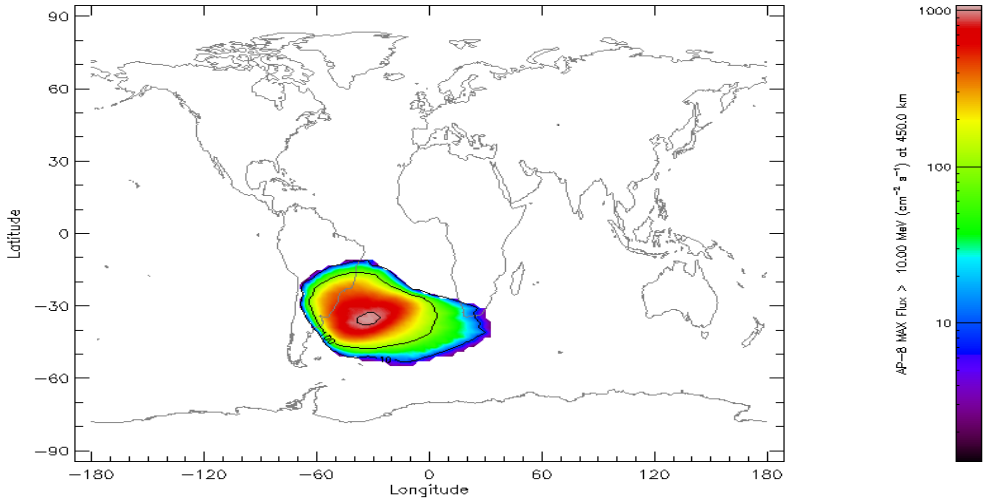


Figure 4-2 - 450km Orbit, >10MeV Proton Flux

In the analysis of the Orbview-2 NASA spacecraft with an orbit of 705 km, the following data was found; with 1440 mils (~37mm) equivalent of shielding, the memory units on the spacecraft was exposed to a flux rate in the 10^2 to 10^3 particles/cm^2-second range [23]. This is similar to the unshielded flux rate CubeSTAR will experience in the SAA.

The 1024 Mbit DRAM modules on Orbview-2 had an average of 170 to 255 SEU/day, which means approximately 1.5-2 byte errors per megabyte per day.

We can assume CubeSTAR will experience some SEUs in radiation sensitive devices such as CMOS registers and SRAM. Since no analysis have been made, it is not possible to estimate the error rate. Other types of SEEs are rare (but not impossible) due to the rapid decrease in flux at higher MeV levels. Each subsystem on CubeSTAR has an overcurrent protection device in order to mitigate the effects from SEL. As a result, SEL and other non-SEUs are not considered when designing each subsystems hardware.

4.5. ORBIT AND BUDGET ANALYSIS

Communication with the UiO Ground Station will be available 3.6% of the time within a 24-hour period. The total amount of informational data possible to download per day is estimated at 1 893 600 byte, or ~7400 packages of 256 byte [24].

The sampling interval for the housekeeping data has been set to 5 minutes during normal operation. Here the OBC will sample full housekeeping information from each subsystem and gather it into a single package of 256 bytes. During normal operation, the OBC will collect 288 packages of 256 bytes per day, which is about 4% of the daily telemetry budget.

When a subsystem is not running as it should, or has events that is of interested, the OBC will receive a notification in the status byte. This means additional information may be stored every 10 seconds. This is limited to the subsystems that are sending the notification and only when the flag state for that subsystem changes. The OBC will not collect redundant sensor data, unless it is asked to do so.

CubeSTAR will be within range of the ground station several times a day; however, it can take up to 10-12 hours between two passes. Because of this, the OBC must be able to store the housekeeping data for as long as possible.

The total amount of memory required for 10 hours is at least 30 KB for the 5-minute sample interval (i.e. 120 diagnostic packages) and some storage may be required for the additional events and housekeeping.

Chapter 5:

Hardware Implementation

5.1. DESIGN CONCIDERATIONS

During the lifetime of the satellite, the OBC and EPS will always be online. Because of this, it is important that these systems consume as little power as possible. In case the batteries reach critically low voltage levels and the satellite goes into power saving mode, these two systems should be able to run for as long as possible from the remaining battery capacity.

The supplied voltages to the board are specified to range between 2.5V and 3.6V. All hardware on the OBC subsystem must be able to operate with this voltage as the only power supply.

It is estimated that the hardware should maintain industrial temperature specifications. The industrial temperature specification range from -40 to 85 degrees Celsius.

The size available on the module slot is fixed, which means all the hardware needs to fit into a limited area. There is little room to add many redundancy or complex hardware solutions. The system needs to remain simple and small.

All electronic devices on the OBC module will eventually break down from radiation exposure. As there is no possibility to repair or replace this hardware, the components needs to remain operational for at least 2 months.

The satellite's clock or time reference needs to be accurate and predictable enough in order to satisfy the spatial resolution for the CubeSTAR payload

measurements. Accuracy and drift needs to be known, so the final characteristics of the time source should be tested and documented.

As a final consideration, the system components should be cheap and easy to get. Extended delivery times and high costs are undesirable. As these components could go out of production, a redesign to fit a new component could delay the final product. Expensive components could provide problems for future funding, again requiring some redesign to fit the new budget.

5.2. ON-BOARD CONTROLLER OVERVIEW

The purpose of the OBC hardware is to provide a system that can run all the software operations that make up the OBC's functions. It needs to provide a non-volatile storage for the satellite diagnostic data. The OBC requires an accurate time or clock reference and the OBC hardware must be able to survive the radiation hazards in space.

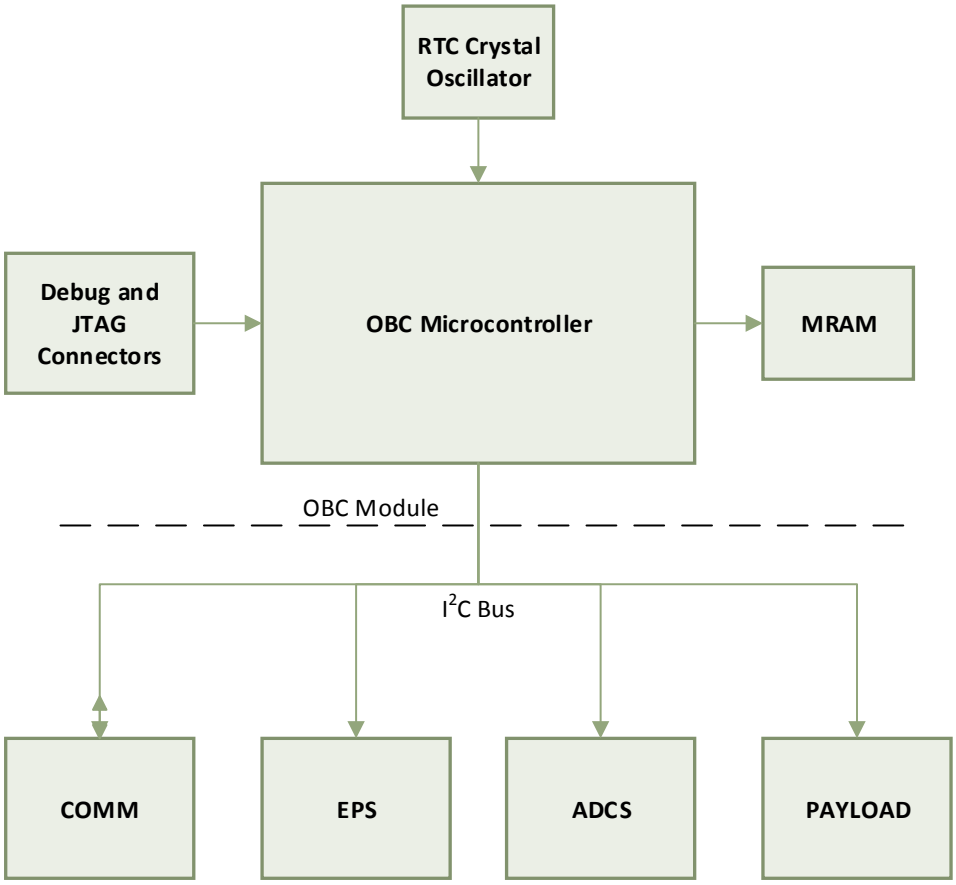


Figure 5-3 - CubeSTAR OBC System Overview

The only interface between the OBC and the other CubeSTAR subsystems is routed through the backplane connector. Through the backplane connector, the OBC module is wired to a I^2C main bus with a spare redundancy bus, two additional I^2C lines with access to the EPS and ADCS sensor values and a GPIO line connected to the EPS.

A debug connector with access to power and ground, all the I^2C lines, the GPIO line and two USARTs (OBC and EPS) are included on the OBC module circuit board. This makes it easier to connect external debugging tools to the satellite during integration and final testing.

"Figure 5-3" shows how the OBC microcontroller is connected to all the elements on the OBC module and subsystems on the satellite. The arrows show in which direction the control or communication is initiated.

5.2.1. A different approach

When looking at how the tasks are divided on the satellite, a commercial design would most likely integrate the OBC, COMM and EPS into a single processor, making a much more compact and simple satellite. This would leave room for a complete standby spare module and other redundancy options, as well as reducing the internal communication in the satellite.

Then why does the OBC have its own hardware? The integrated method produces some problems for a student driven project, as it is difficult to maintain and develop the work that has been done previously. Once a student has finished his part of the project and is no longer participating in the development, the future changes and integration work will become difficult. Documentation is often limited and there is not much top-down project management. Once you add more students to the system and they finish their work, the software will soon become unmanageable.

When you consider the different schedules of the students, preferences in development tools, languages and design choices, the most education value gained from a student project is achieved by splitting it into different hardware segments with their own software.

5.2.2. Earlier work

At the beginning of the thesis work, all the satellite subsystems were using the ATXMEGA128A1 microcontroller, except the payload. The OBC was a

controller on the backplane of the satellite, intended to measure sensor data and maintain a healthy power operation and temperature range in the satellite. The EPS was a separate subsystem, using two microcontrollers for maximum power point tracking [25] and the ADCS was partially implemented on a microcontroller with de-tumbling [26].

Several things were changed since the initial designs were implemented. A new IC with built in power point tracking was found, replacing the old EPS subsystem. The OBC was moved to a subsystem slot because it required some space for its own hardware. This opened up for the backplane processor to become a part of the new EPS, integrated into the backplane. The ADCS system was changed to a FPGA solution, instead of a microcontroller.

5.3. MICROCONTROLLER

The OBC uses an Atmel XMEGA microcontroller. This microcontroller can operate from 1.8 V to 3.6 V, ultra-low power consumption using sleep modes and industrial temperature specifications with temperature calibrated system clock. It has support for a large memory and it has an integrated RTC function. It is cheap and requires very few external components, which means there will be little hardware complexity.

No radiation characteristics are found on the XMEGA, but some radiation tests were performed on a similar device from Atmel; the ATmega128.

The SEU error rate for the ATmega128 SRAM has been found to be approximately 5.4×10^{-13} SEEs per byte and proton/cm^2 [27]. We then find that 12 KB SRAM gives ~6.5×10^{-9} SEE per 12 KB SRAM and proton/cm^2.

Since we do not have any data on the SEU sensitivity in the XMEGA's SRAM, the only alternative would be to assume the XMEGA has similar characteristics to the ATmega128. We can then approximate the chance of a SEU for each second CubeSTAR is within the SAA by using the proton flux rate. The calculations for this will not be performed in this thesis, but could be done in future work if desirable.

5.4. CRYSTAL OSCILLATOR

The crystal oscillator is intended to provide holdover accuracy between synchronization of the satellite clock. In order to have as much predictability

as possible, the crystal needs very good frequency stability or predictable characteristics. The scientific instrument in the satellite ideally wants a time accuracy that allows for mapping of scientific data down to a 50 or 100-meter grid. This means the tolerance and drift cannot be more than 5 to 15 ms at any given time.

1 ppm equals 1 µs drift per second. If the clock has an average of 11.57 ppm drift, this will result in 1-second drift per day.

In "Table 5-4 - Oscillator Types" the most common crystal and oscillator sources are compared [28]. This overview gives a general idea which type of clock type would be most suitable for the OBC.

Table 5-4 - Oscillator Types

Clock Type	Frequency Tolerance		Temp Stability	Aging /year	Drift/day	Cost
RC Oscillator	1%	$\pm 10 \times 10^{-3}$			15 min	Free
Typical Crystal	50 ppm	$\pm 50 \times 10^{-6}$	$\pm 50\ ppm$	3-5 ppm	5 sec	$
Watch Crystal	20 ppm	$\pm 20 \times 10^{-6}$	$-0.04\ ppm/°C^2$	3-5 ppm	2 sec	$
TCXO/MCXO	0.5 ppm	$\pm 500 \times 10^{-9}$	$\pm 0.5\ ppm$	1 ppm	50 ms	$$
OCXO	1 ppb	$\pm 1 \times 10^{-9}$			100 µs	$$$
Atomic Clock	1 ppt	$\pm 1 \times 10^{-12}$			100 ns	$$$$$

The power consumption of OCXO and atomic clocks are typically higher than 1 W. TCXO has power consumption of a few mW, ceramic and typical quartz crystals consume anything from 5 to 70 µW and watch crystals in the 20 to 50 nW range.

The TCXO has the most favorable characteristics due to the high precision. These are unfortunately not ideal for this application. The cost and power consumption for a TCXO is relatively high. It requires a stabile power source and it is difficult to get in 32768 Hz frequency. A TCXO is just a factory-calibrated oscillator using a standard crystal and a look-up table. Slightly better characteristics can be achieved either with a MCXO, or simply by post-processing.

Instead of increasing the complexity and cost of the board, a tuning fork with post-processing is the optional choice. A tuning fork watch crystal has a very predictable temperature characteristic and this allows very good accuracy in post-processing. It follows the specifications of the RTC in the XMEGA and uses very little power. A good tuning fork crystal is the CM315-32.768KEZF-

UT. This crystal has a frequency tolerance of 10 ppm and a temperature coefficient of −0.034±0.006 ppm/°C[2]. It is recommended for use by Atmel with the XMEGA in their "AVR4100" application note and by Energy Micro for their EFM32 microcontroller in their "AN0016" application note.

Frequency in a tuning fork crystal can be found using:

$$f = f_0(1 - PPM \times (T - T_0)^2)^2$$

Where f is the frequency at temperature T, f_0 equals the nominal frequency at temperature T_0 and ppm is the crystal temperature coefficient. Using this for the CM315, we can find a frequency curve by temperature as shown in "Figure 5-4 - Crystal Frequency by Temperature".

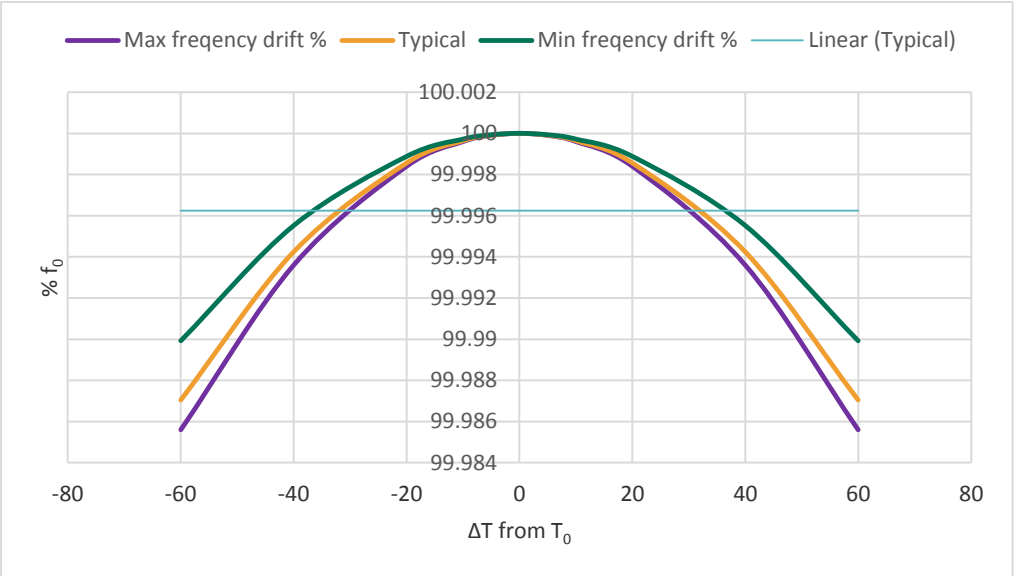


Figure 5-4 - Crystal Frequency by Temperature

The nominal temperature and frequency is unknown and varies between different crystals. The turnover temperature for the CM315-32.768KEZF-UT is $25°C \pm 5°C$ and the nominal frequency is 32768 Hz with a tolerance of $\pm 10\ ppm$. Other factors such as capacitive load, vibration, magnetic and electric fields, radiation and aging can affect the frequency as well.

You can make a good estimation if you find the average frequency and average temperature, then match that towards the temperature coefficient formula. Because there is two unknowns and only a single equation, one

[2] Corrected mistake in formula provided Atmel in AVR4100 p.7 Rev. 8333D-AVR-07/11

could check the nominal frequency using both 20℃ and 30℃ turnover temperature and select a reasonable value between these two.

The frequency of a crystal (F_S) with capacitive load (F_L) is found by [29] [30]:

$$F_S = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

$$F_L \approx F_S \times (1 + \frac{C_1}{2(C_0 + C_L)})$$

For CM315 C_0 is ~0.95 pF and C_L is 12.5 pF. By looking at various capacitance ratios (r) in the 200 to 1000 range, (e.g. $C_1 = \frac{C_0}{r} = \frac{C_0}{1000} = \sim 0.00095\ pF$), we can estimate the change in frequency ppm/pF of C_L.

$$for\ r = 1000 \rightarrow 1 + \frac{0.00095}{2(0.95 + 12.5)} = 1 + 35.3 \times 10^{-6}$$

$$for\ r = 200 \rightarrow 1 + \frac{0.00475}{2(0.95 + 12.5)} = 1 + 176.6 \times 10^{-6}$$

We then change C_L by 1 pF:

$$for\ r = 1000 \rightarrow 1 + \frac{0.00095}{2(0.95 + 11.5)} = 1 + 38.1 \times 10^{-6}$$

$$for\ r = 200 \rightarrow 1 + \frac{0.00475}{2(0.95 + 11.5)} = 1 + 190.8 \times 10^{-6}$$

The ratio between the two frequencies are then:

$$for\ r = 1000 \rightarrow \frac{1 + (38.1 \times 10^{-6})}{1 + (35.3 \times 10^{-6})} = 1 + \mathbf{2.8 \times 10^{-6}}$$

$$for\ r = 200 \rightarrow \frac{1 + (190.8 \times 10^{-6})}{1 + (176.6 \times 10^{-6})} = 1 + \mathbf{14,2 \times 10^{-6}}$$

The calculations show there will be about 3-14 ppm/pF change in C_L, depending on crystal capacitance ratio. A tuning fork tends to be at the lower end of the scale.

Then how accurate does C_L actually get and how much will it change with temperature or other factors?

The capacitive load as show in “Figure 5-5” is calculated with the following formula:

$$\sum C_L = \frac{(C_{L1} + C_{P1} + C_{EL1}) \times (C_{L2} + C_{P2} + C_{EL2})}{C_{L1} + C_{L2} + C_{P1} + C_{P2} + C_{EL1} + C_{EL2}}$$

Alternatively, a simpler formula, where C_{stray} is about 2-5 pF:

$$\sum C_L = \frac{C_{EL1} \times C_{EL2}}{C_{EL1} + C_{EL2}} + \frac{C_{stray}}{2} \leftrightarrow C_{EL1} = C_{EL2} = (2 \times C_L) - C_{stray}$$

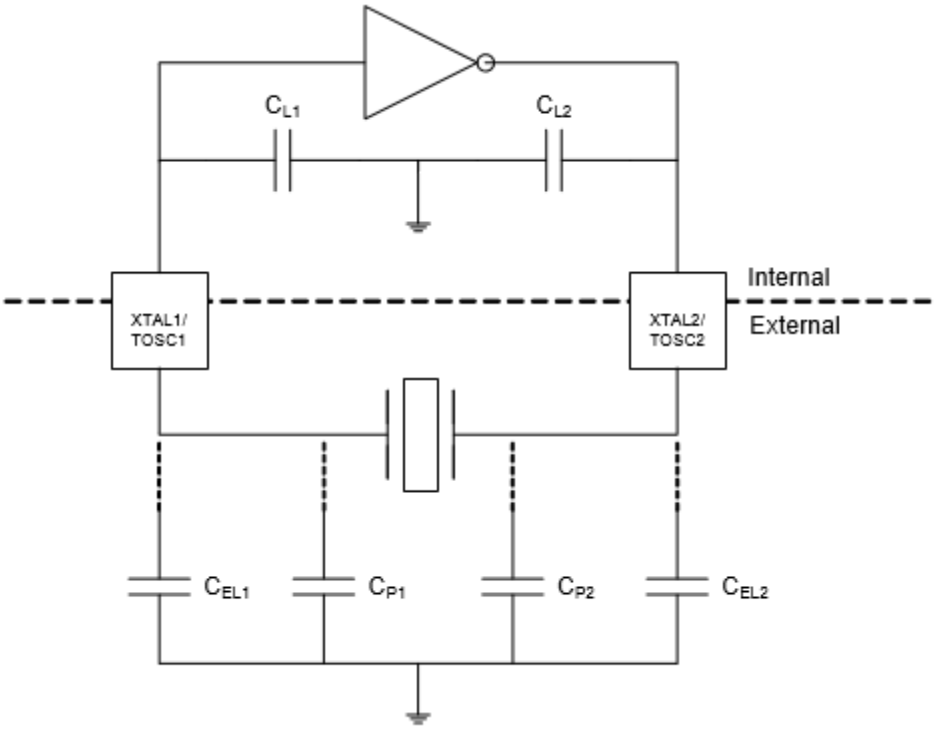


Figure 5-5 - TOSC Capacitive Load

Using the simple formula, we get 20 to 23 pF for each of the external capacitors. The closest value in the E6 series is 22 pF, which is a typical value used with the XMEGA. Let us investigate and see the difference by using the more advanced formula.

C_{P1} and C_{P2} comes from parasitic capacitance in the PCB (1-3 pF). C_{L1} and C_{L2} are the parasitic capacitance in the package and pins of the microcontroller

(ATXMEGAA1 IBIS shows 4.71 pF total and ATXMEGAA1U shows typical values of 4.0 and 4.1) [31].

Using the 22 pF capacitors we got from the previous formula, we get the following C_L:

$$\frac{(4.71 + 1 + 22)^2}{2 \times (4.71 + 1 + 22)} = 13.855\ pF$$

Since we want C_L to be 12.5 pF, as specified by the CM315 datasheet, we can refactor the formula and see what the ideal capacitor value is. Since all the capacitance values are equal, the formula becomes:

$$\frac{4.71}{2} + \frac{1}{2} + \frac{C_{EL}}{2} = 12.5 \rightarrow C_{EL} = 25 - 5.71 = 19.29\ pF$$

The difference between the formulas is very small and the parasitic capacitance is hard to predict. For the ATXMEGAA1 and ATXMEGAA1U, the simple formula gives the same E6 series capacitor value. Other microcontrollers will have different characteristics and some have much larger pin capacitances (even up to 12-18 pF). An 18 pF capacitor would make C_L = 11.855 and 20 pF makes C_L = 12.855. Using an 18 pF capacitor means the pin and stray capacitances could be anywhere between 5 and 7 pF and leave the C_L error margin with less than 1 pF.

As the C_L error margin could be within ~0-2 pF, this gives an uncertainty range of anything from 0 to 30 ppm in crystal frequency based on the crystal and capacitor value. A capacitor with tolerance of 10% would add another 1 pF to the C_L error margin. Without measuring the finished PCB with a high precision capacimeter, trying to achieve a consistent nominal frequency within 5 ppm of the crystal specification is difficult. This is in addition to the ppm grade of the crystal. This means the focus should be on temperature stability and predictability rather than frequency tolerance, as the tolerance alone would typically leave a drift of no less than ~500 ms per day.

In order to avoid a large variation with temperature, the external capacitors needs to be an EIA Class 1 dielectric, specifically a C0G/NP0 capacitor. These have a temperature stability rated at $\pm 30\ ppm/°C$ and will not drift more than ~0.1% across the entire temperature range. The most ideal capacitor

choice would then be an NP0 with 18 pF and a 1% or better tolerance; however, both 15 pF and 22 pF would be acceptable.

The temperature drift of the capacitor gives ~0.03-0.3 ppm drift in the crystal frequency over the entire temperature range.

5.5. MEMORY

5.5.1. Memory Interfaces

There are several types of interfaces available for memories and the most dominant ones are I^2C, SPI and Parallel. I^2C is very convenient because this interface type is already used on the satellite and there is no need to write new software drivers to use it. SPI tends to be slightly faster than I^2C, but there is no other advantage of using this interface over I^2C. The parallel interface is very fast, but requires significant amount of IO pins. You can access an address field up to every clock cycle.

The ATXMEGAA1 has an External Bus Interface (EBI) that allows the microcontroller to access external peripherals or memory with a parallel interface directly using its internal memory bus. This allows up to 16 MB of address space, using 24-bit addresses. The EBI makes software complexity significantly lower, as you can access all the memory simply by using an address pointer. Because the address pointers (K, X, Y and Z registers) in the ATXMEGAA1 are only 16-bit, RAMP pointer registers are intended to cover the remaining eight address bits and with the GCC compilers must be manually written when accessing addresses above 64 KB. See “6.2.1 Programing Language” for more detail on compiler options.

When comparing these interfaces, the parallel interface is the ideal choice because of the speed and easy integration. The memory should have an 8-bit data width, as this would be most compatible with the XMEGA EBI.

5.5.2. Memory Technologies

There are multiple technologies to choose from when selecting a memory. The most common storage mediums on satellites are DRAM, SRAM, Flash and EEPROM [32]. Each memory type is generally used for specific applications.

- EEPROM is used for used for small amounts of startup data, like configuration, calibration and debug information.

- Flash is a type of EEPROM with faster erase capability. This memory is used for data similar to the EEPROM. The benefit is that it can store larger amounts of data, like full device applications, and data storage.
- SRAM is a typical runtime memory for a processor and is used for buffers and temporary variables.
- DRAM is generally used for data processing applications and large buffers of data.

Table 5-5 - Comparison between standard memory technologies

Technology	Typical Capacity	Power Usage	Access Time	Rewrite Cycles	Data Retention	SEU Tolerance
DRAM	<1 GB	High	~50 ns	∞	None	Low
SRAM	<8 MB	Medium	~10 ns	∞	None	Low
Flash	<1 GB	Low	~70 ns	5k-50k	20 years	High
EEPROM	<32 KB	Low	~1 ms	1 million	10 years	High

In “Table 5-5”, we can see that all these memories have some disadvantages when it comes to the OBC and certain methods are needed to circumvent these. Due to the limited SEU tolerance for DRAM and SRAM, some type of EDAC method should be implemented when using these memories. Because of the limited rewrite cycles and high power consumption when writing to Flash, this type of memory should preferably only be used with bulk operations.

As an alternative to these memories, two new technologies have recently emerged, FeRAM and MRAM. As shown in “Table 5-6”, they both have very favorable characteristics for the OBC application. FeRAM and MRAM do not require a charge pump to write to memory and as such requires a lot less power. The memory cell is not subject to SEUs, because the storage element does not contain an electric charge. They both have much higher endurance than Flash and EEPROM and have faster access time.

Table 5-6 - Comparison of FeRAM and MRAM

Technology	Typical Capacity	Power Usage	Access Time	Rewrite Cycles	Data Retention	SEU Tolerance
FeRAM	<512 KB	Low	~60 ns	∞	10 years	High
MRAM	<512 KB	Low	~35 ns	∞	20 years	High

The OBC memory would be used a lot during run-time, so Flash and EEPROM

are not desirable. Neither is the density in DRAM, as SRAM has enough capacity. This makes the ideal candidates FeRAM, MRAM or SRAM.

5.5.3. Memory Choice

As seen in the budget analysis, the required memory for the OBC is at least 512 Kbit for housekeeping data, but it would be convenient to be able to add additional memory in case it is needed later in the project.

When looking at all the criteria, we find some good alternatives:

- 1 Mbit (128 KB) FeRAM from Ramtron.
- 4 Mbit (512 KB) MRAM from Everspin.
- 16 Mbit (2 MB) Rad-Hard/EDAC SRAM from Texas Instruments (4 Mbit w/8 bit data width)
- 16 Mbit (2 MB) Rad-Hard/EDAC MRAM from Aeroflex

Table 5-7 - Comparison of various memory devices

Brand	Name	Technology	TID*	LETth*	SEE*	Availability	Price
Texas Instruments	SMV512K32-SP	CMOS SRAM	3e5 rad(Si)	110 MeV cm^2/mg	5e-17 No SEL	No Stock	50$
Ramtron	FM28V100	FeRAM	N/A	20 MeV cm^2/mg	Low	COTS	15$
Everspin	MR2A08A	MRAM	7e4 rad(Si)	17.4 Mev cm^2/mg	No SEU	COTS	15$
AeroFlex	UT8MR2M8	MRAM	1e5 rad(Si)	100 MeV cm^2/mg	No SEU No SEL	On demand	$$$$

* The Ramtron and Everspin numbers are based on tests on same technology, but not device.

Based on the comparison done in “Table 5-7”, the AeroFlex UT8MR2M8 has by far the best characteristics. There would be no radiation problem for the memory during the entire CubeSTAR mission and the memory would have more than enough capacity. However, the high price and long delivery time makes this a difficult choice.

Texas Instruments SMV512K32-SP is a decent choice, as any radiation problem would be very unlikely. It is volatile and a backup battery solution would be required to achieve the desired functionality. As this is not a COTS item, the delivery time and minimum quantities could be a problem.

Everspins MR2A08A is the 4 Mbit device in their parallel MRAM series. An optional 16 Mbit device exists with pin-swap compatibility making storage space a non-issue. As MRAM is SEU immune and the device has high TID, any radiation problem would be unlikely.

Ramtron FM28V100 was the highest density FeRAM parallel memory with 8-bit data width, but has only 1 Mbit capacity. This should be enough, but it would limit expansions or additions to storage later in the project. There is limited testing or documentation on radiation effects for this device.

Everspins MRAM and Ramtrons FeRAM are considered the best choices due to radiation characteristics, availability and cost. There is a lot more documentation regarding radiation testing with MRAM compared to FeRAM. One of the few tests on FeRAM showed it is possible for FeRAM to get errors in memory from heavy ion radiation.

The FeRAM cells are immune to SEUs, however the CMOS control logic is not and it is assumed this is the cause for the errors in memory observed during radiation testing [33] [34] [35].

MRAM has been used in previous space missions, like the Japanese SpriteSat. [36] Lastly, MRAM was picked for further rad-hard development by AeroFlex due to its characteristics. In total, this makes Everspins MRAM a more desirable choice over Ramtrons FeRAM, especially due to the TID and SEU characteristics.

5.5.4. Radiation problems

There is some conflicting information regarding the Everspins LETth. A test performed by Aeroflex on the 16 Mbit device showed a SEL threshold at 17.4 MeV cm^2/mg [37]. This means the CMOS logic within the device could experience SELs from particles with LET levels above 17.4 MeV cm^2/mg. A different test performed on the 1 Mbit device shows a LETth of 84 MeV cm^2/mg [38] [39].

Everspin support mentioned both the 130 nm and 180 nm processes were used in production (as of 2012-03-29). They did not provide any specific details as what components were made with the 130 nm process, but the MR2A08A was manufactured using the 180 nm process.

The recommended SEL limit for Hinode/Solar-B, which has up to almost 700 km altitude in its orbit was set at 37 MeV cm^2/mg [40]. If the 17.4 MeV cm^2/mg characteristics are correct for the MR2A08A, there is a possibility for latch-up in the device.

On the older generation MRAM from Freescale Semiconductors, the worst case Galactic Cosmic Ray induced SEL rate was estimated to be approximately 1/4000 device-day [41]. The MRAM in CubeSTAR will be powered on less than 10% of the flight time. If we assume the SEL rate is similar between the two devices, this means there is less than 1% chance of a Galactic Cosmic Ray induced SEL during a 1-year mission.

5.5.5. Power Regulator

The MRAM operates within 3 and 3.6 V. This means the MRAM requires a power regulator in order to work down to the specified 2.5 V. The regulator must have an input range of at least 2.5 V to 3.6 V and a fixed output of 3.3 V. To simplify the connection between the microcontroller and the memory, the entire OBC board will be regulated. In order to supply the entire OBC, the power regulator needs to provide a minimum of 200 mA peak current and the power dissipation must be low enough for use on the satellite. The power regulators shown in “Table 5-8” provide the necessary functionality.

Table 5-8 - Comparison of various power regulators

Brand	Name	Input Range	Peak Out	Efficiency	Layout	Availability
Linear Technology	LTC3531-3.3	1.8V to 5.5V	<200 mA	<90%	Easy	Low
Texas Instruments	TPS63001	1.8V to 5.5V	<800 mA	<96%	Easy	High
Texas Instruments	TPS61252	2.3V to 6.0V	<1500mA	<92%	Medium	High
Texas Instruments	TPS61070	0.9V to 5.5V	<300 mA	<90%	Medium	High

Texas Instrument TPS63001 was picked based on general availability, easy board layout and good power output margin compared to alternative components.

Based on the datasheet, the inductor value for TPS63001 was calculated using these formulas:

Minimum L1 step down

$$L_1 = \frac{V_{OUT} \times (V_{IN1} - V_{OUT})}{V_{IN1} \times f \times 0x3A} = \frac{3.3\ V \times 0.3\ V}{3.6\ V \times 1250\ kHz \times 0.3\ A} = 0.73\ \mu H$$

Minimum L2 boost

$$L_2 = \frac{V_{IN2} \times (V_{OUT} - V_{IN2})}{V_{OUT} \times f \times 0x3\ A} = \frac{2.5\ V \times 0.8\ V}{3.3\ V \times 1250\ kHz \times 0.3\ A} = 1.62\ \mu H$$

1.5 µH was chosen as the most reasonable value.

5.5.6. Power-Distribution Switch

To reduce the static power consumption from the MRAM, a shutdown function for the memory is implemented. In order to simplify the satellite component list, the OBC uses the TPS2557 power-distribution switch that is used in the rest of the satellite.

5.6. PCB REALIZATION

Three PCBs have been designed and manufactured, but only two of them have been tested and verified with components. The initial design was a prototype that included some hardware features that were never used. This included a 12 MHz crystal, an additional processor, external I^2C memory and watchdog in hardware. Ver. 1.1.0 removed all these components and the board was feature complete and bug free. However, after the hardware for the OBC was complete, two things changed:

The ATXMEGA128A1U microcontroller was released on the market, which was pin-swappable with the older ATXMEGAA1. The A1U had several improvements like built-in CRC generator, simplified EBI interface and more flexible configuration options than the older A1. They could both run the same software.

The PCB size for the CubeSTAR subsystems were changed, so Ver. 1.1.0 of the board would no longer fit in the CubeSTAR satellite. This meant a new layout had to be made eventually.

Since Ver. 1.1.0 did not take advantage of the improved EBI interface of the A1U microprocessor, a new layout was made, Ver. 1.2.0. This had the new module size, and took advantage of the simpler layout for the ATXMEGAA1U EBI. This board was not built or tested with components due to time constraints on the thesis.

Schematics and layout of the PCBs was done using the Zuken CADSTAR v13 suite. All the boards were designed as four layer, with power and ground in the two middle layers.

“Figure 5-6” shows Ver. 1.1.0 of the board with all the components mounted. It includes the ATXMEGAA1U microcontroller, the CM315 crystal, the MRAM memory with address latches, and the power circuitry components.

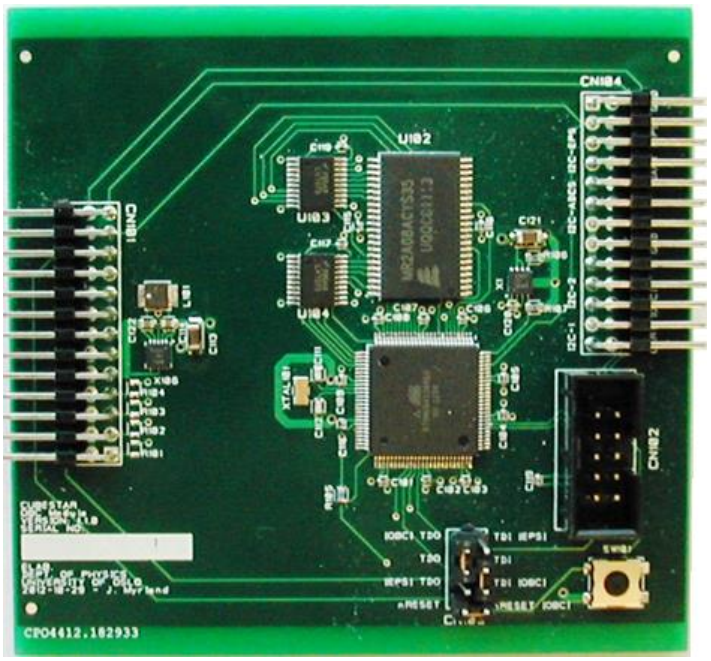


Figure 5-6 - OBC Module Ver. 1.1.0

Chapter 6:

Software Implementation

6.1. FLOW AND MAIN FUNCTIONS

The OBC will be in a sleep state during most of its operation and an interrupt source is required to wake the processor. Every ten seconds an RTC interrupt will wake the processor and initiate status collection. Functions such as beacon and gathering of statistical diagnostic data is activated at their given intervals. The timeline of these events are shown in “Figure 6-7 - RTC Tick Timeline”.

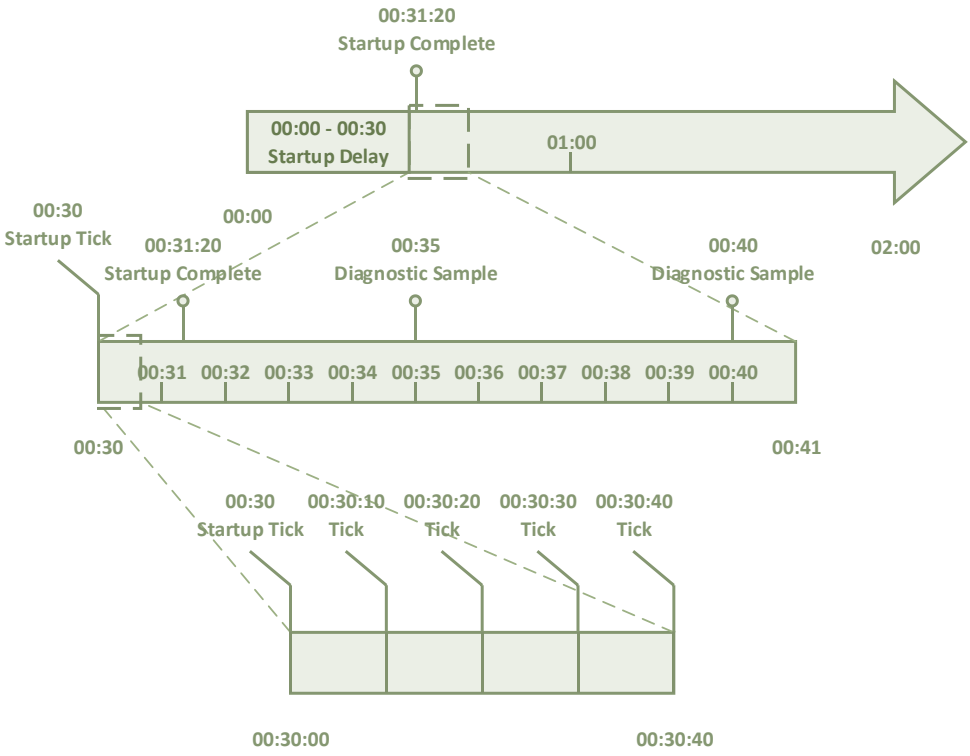


Figure 6-7 - RTC Tick Timeline

Once the interrupt is complete, the controller will start its main program loop. First, the controller will check if the status flag collection is enabled. If the status flag collection is enabled, the controller will start gathering a status byte from all subsystems. The status byte contains flags that will tell if there is something of interest, or an event that needs attention within the satellite. Some of these flags can trigger further diagnostic data to be downloaded from a specific subsystem. All state changes on these status flags will be logged in memory.

Once the overall satellite status is checked, the system will download diagnostic data if it was enabled by the state of a status flag, or an interrupt source. This data will be time stamped and saved in the memory.

If a flag was enabled, the controller will begin its event handling and respond appropriately. Flags can be enabled by a status byte, from an IRQ line, an I^2C command or from an error in the OBC software. Once the main program loop is complete, the controller will enter a sleep state until the next interrupt. This flow of operation is shown in “Figure 6-8 - Flowchart OBC”.

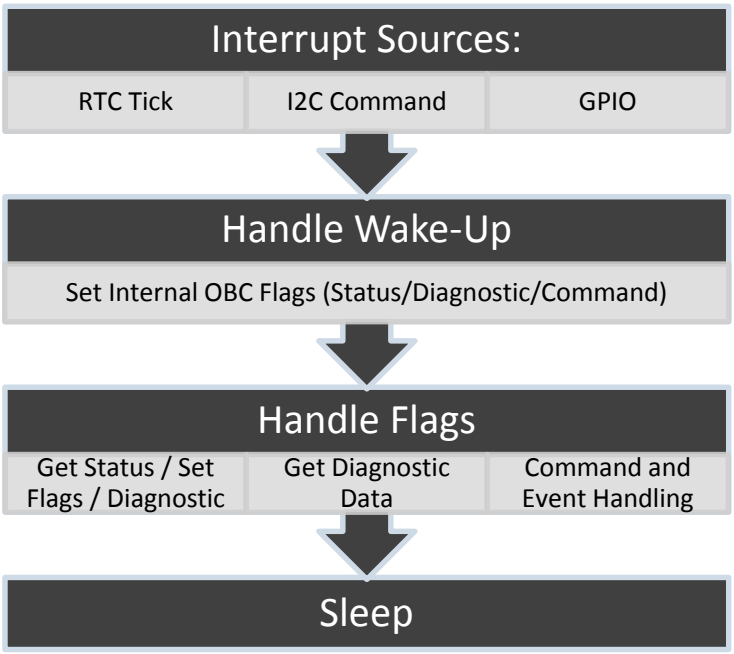


Figure 6-8 - Flowchart OBC

6.1.1. Status flags

The ADCS, COMM and EPS each have an error notification flag and an event notification flag. If the controller reads an error flag, diagnostic data will be collected from the specific subsystem and stored in memory. The subsystem will then be reset by an I^2C command. If the event notification flag is set, the controller will collect a limited diagnostic package from the subsystem and store it in memory.

From the EPS, there is a power warning flag and a flag indicating one or more subsystems are offline. The ADCS has a flag indicating if the satellite attitude is within the payload specification parameters and a flag indicating if it is in power save mode, or in normal operation. The COMM system responds with a byte value of 0x00, which will to notify the OBC it is running as normal.

6.1.2. Diagnostic data

A diagnostic package contains a combination of sensor data and status information. Diagnostic data will be collected every 5 minutes, or when the OBC receives a notification from a subsystem through the status byte. The diagnostic data will only be collected from online subsystems.

Total size of the diagnostic package is 256 bytes and contains data from the ADCS, COMM, EPS and OBC subsystems, as well as eight CRC bytes. The data will include all information about the satellite's health and operation as determined by the individual subsystems.

6.1.3. AI logic

The flag-handler logic is the "intelligent" part of the software and enables the satellite to be autonomous. It will check status flags, e.g. which subsystems are online and if the payload probes are deployed. This part of the software performs all tasks required by the OBC. This includes operations like command requests, time synchronization, payload collection and so on. A specific structure and crosschecking scheme is implemented so that tasks are performed by priority and in proper sequence.

6.2. SOFTWARE REQUIREMENTS

A complete I^2C Master and I^2C Slave driver needs to be implemented in the software. All data handling should be done in the I^2C Master driver directly and no further data shuffling or processing should be done once the transaction is complete. This will limit the complexity while integrating the driver with the software, but could leave parts of the driver very application specific. The slave driver needs to respond only on OBC address match and reject all command values that are not valid. These commands are explained in detail in the appendix. Both drivers needs to use 16-bit variables for the transaction length.

The memory interface should be as transparent in the code as possible. It needs to work with the I^2C driver, while keeping transaction time as low as possible.

In order to keep an exact time on board the satellite, a precision RTC Clock needs to be implemented. Some method of synchronization and compensation for drift in the clock is required. Synchronization must be logged, with details about updated time value and drift.

Due to the XMEGAs SRAM being vulnerable to SEU, it is important to pay special attention to all variables, i.e. assume either one of them could experience a bit flip. All commands and states need to be verified as valid hamming code. Logic cases should consider potential bit flips in order to prevent possible deadlocks.

Optional: A watchdog timer could be implemented to prevent deadlocks. This would require a predictable loop-time and short enough wake-up cycles. The risk of a deadlock needs to be evaluated based on software structure and what other methods of recovery exists in the system.

Optional: A master and slave boot loader for the XMEGA firmware could be implemented to increase reliability. This would remove the chance of a SEU in the satellite's application firmware and give flexibility regarding software upgrades. If the firmware becomes corrupted and the system has no boot loader, there is no chance for recovery. Even with a boot loader, the boot loader section would still be vulnerable to firmware corruption.

6.2.1. Programing Language

There are three different programming languages available for the AVR XMEGA and then some additional compiler options.

Assembly is a translation of machine level code into readable assembly commands. When writing assembly code, every processor action is specified and knowledge of the architecture is required. This type of code tends to be more efficient and optimized, but is difficult to manage when complexity and size of the program increases.

C is often called a middle level language, because it requires some control over low-level operations and there are little to no restrictions regarding memory access from anywhere within the code. However, it simplifies the code into more human friendly text, e.g. logical and arithmetical operators that are not found in assembly and it can perform advanced machine code operations using very few statements.

C++ takes advantage of object-oriented programming and other enhancements to C, while containing all the benefits of the C language. It allows many different programming styles and is more flexible than C in this regard. A C application will most likely work directly with a C++ compiler or with only a few minor tweaks to code.

The software was initially written using the AVR-GCC C compiler. The alternative to C was assembly and that was not considered an option. Later in the project C++ compilers were released for the XMEGA and it was decided to modify the code to AVR-GCC C++ in order to take advantage of the more advanced compiler features. After some tweaking of the C code, the compiled C++ code had a large reduction in size. This was most likely due the global variable declarations used with the C compiler. The software for the OBC was kept very true to C syntax and porting back to C should not be difficult.

Optional compilers include the IAR compilers and CodeVision. The other subsystems on CubeSTAR wrote their software using GCC, so there was little reason to consider a different compiler.

6.3. HAMMING CODE

As hamming codes are used in both the OBC software logic and the I^2C bus protocol, we need to find a set of hamming codes that can be used.

Since the XMEGA is an 8 bit microcontroller, it would make sense to find a data width where $(2^Y+Y+1) <= 8$. As found in "3.2.3 Hamming Code", Y = 2 gives us a bit width of seven. This happens to be the well-known hamming (7, 4)-code where there are four data bits and three check bits.

To take advantage of the full 8-bit bus width, we can add a parity bit to the most significant bit of the hamming code value. Using the example value from the theory section, we can add a parity bit to the hamming code and a data value of 0001 then gives us 0b10000111, or 0x87. This means the values will get a hamming distance of four. Using this method, we can find the rest of the hamming values and insert them into "Table 6-9 - Hamming (7, 4) values w/parity".

Table 6-9 - Hamming (7, 4) values w/parity

Data value	7 (P0)	6 (D3)	5 (D2)	4 (D1)	3 (C2)	2 (D0)	1 (C1)	0 (C0)	Hamming value
0000	0	0	0	0	0	0	0	0	0x00
0001	1	0	0	0	0	1	1	1	0x87
0010	1	0	0	1	1	0	0	1	0x99
0011	0	0	0	1	1	1	1	0	0x1E
0100	1	0	1	0	1	0	1	0	0xAA
0101	0	0	1	0	1	1	0	1	0x2D
0110	0	0	1	1	0	0	1	1	0x33
0111	1	0	1	1	0	1	0	0	0xB4
1000	0	1	0	0	1	0	1	1	0x4B
1001	1	1	0	0	1	1	0	0	0xCC
1010	1	1	0	1	0	0	1	0	0xD2
1011	0	1	0	1	0	1	0	1	0x55
1100	1	1	1	0	0	0	0	1	0xE1
1101	0	1	1	0	0	1	1	0	0x66
1110	0	1	1	1	1	0	0	0	0x78
1111	1	1	1	1	1	1	1	1	0xFF

We now take these hamming values and customize them to our needs. By sorting the values found in “Table 6-9” by ascending order, we define a new set of data to correspond to the various hamming values as shown in “Table 6-10”. These values are then used for data encoding, flags and states for the OBC and commands for the I²C bus protocol.

Table 6-10 - CubeSTAR Hamming Code List

Code	0x00	0x1E	0x2D	0x33	0x4B	0x55	0x66	0x78
Data	0	1	2	3	4	5	6	7
Code	0x87	0x99	0xAA	0xB4	0xCC	0xD2	0xE1	0xFF
Data	8	9	A	B	C	D	E	F

Adding the parity bit to a different bit position gives you alternative value combinations, however this particular combination has some convenient benefits. First, it is very easy to convert between hamming value and data value, since the data value is already presented in the top 4 bits. You can find the hamming value without much work if you have the data value; when the top 4 bits have an even number of 1’s, the bottom 4 bits are identical and when the top 4 bits have an odd number of 1’s the bottom 4 bits equals the inverted value of the top 4 bits.

6.4. I²C BUS COMMUNICATION

6.4.1. I²C Drivers

The I²C drivers were written specifically for the I²C Bus Protocol. They allow many low-level operations to be transparent to the application. This includes rejecting all unknown commands, upload and download of data directly to memory and overflow protection. It can be used in both polling and interrupt based operation, which allows e.g. a boot loader to use it without moving the interrupt vector table. It will work across many I²C interfaces simultaneously, but will only run a single I²C master and slave operation at a time. Because there are no buffers in the driver by default, it only requires 26 bytes in memory for the complete I²C Master and Slave functionality.

6.4.2. I²C Bus Protocol

CubeSTAR’s main bus protocol follows the standard Philips I²C specification. A master request requires an address match on one of the slaves on the bus

before it can start transferring data. The first data byte is a command byte. If the command byte does not match the slaves command list, the slave will cancel the transaction.

All command bytes have a hamming distance of four with each other, which means the command byte needs exactly 50% (4 bit), 75% (6 bit) or 100% (8 bit) data corruption before a wrong command could be sent. If the command byte is scrambled or has a random value, it would be less than 5.5% chance for the value to match one in the command list. A command list has a maximum of 14 commands.

$$Error\ \% = \frac{Command\ List\ Length}{256} \times 100\%$$

The bus protocol allows any subsystem to send a command to any other subsystem, as there is no indication of who sends a message to the receiver. The implementation limits the bus to the OBC and COMM systems only. In case the OBC stops working, this allows the ground station to send commands to any subsystem through COMM. During normal operation, COMM can only send commands to the OBC.

6.4.3. I²C Commands

All CubeSTAR I²C commands are described in detail in “Appendix A”.

The OBC will sometimes receive commands from the ground station. This will happen when the ground station crew wants the OBC to perform a task beyond its routine operations. These are the commands that can be sent to the OBC:

- Run a CRC check on the flash application memory, and compare to a static variable in the MRAM data storage.
- Perform a soft restart of the OBC.
- Upload of data to the MRAM memory.
- Initiation of housekeeping or telemetry data bursts.
- A time update command.

All commands are handled in the flag-handler logic in the software. These are performed after the routine data logging operations are completed by the OBC.

Commands from the ground station, or events in the satellite can trigger the OBC to send additional commands to the internal subsystems. This happens in particular during the startup sequence of the satellite, where many sequential operations needs to be performed.

- Deployment of antenna and probes.
- Subsystem power-on.
- Uptime and clock synchronization.

E.g. during startup and deployment of the Langmuir probes, the OBC will send a “Release m-NLP” command to the EPS subsystem. This command transaction sends two bytes and receives one byte:

- The first byte, command value, is equal to 0xE1.
- The second byte is a hamming code that identifies which of the four probes should be deployed.
- The third byte, which is the response byte, is a hamming code identifying which probes are already deployed.

The principle of this transaction remains the same for the other commands.

6.4.4. Beacon Content

After diagnostic collection from each of the subsystems is completed, the OBC will write its own data to the beacon package. The content of the OBC diagnostic data includes:

- The on-board time in number of weeks and 10-second ticks.
- Subsystem and I^2C status, error and states.
- Amount of data packages in memory.

Once all the data is written, a CRC generation sequence is run. The OBC takes advantage of an internal CRC generation module in the ATXMEGAA1U microcontroller. This creates 16-bit CRC values that are added to the end of each 62 byte data segments in the beacon package. In total 256 bytes.

6.4.5. Error handling on the I^2C bus

If an I^2C transaction fails, an I^2C error flag will be set for that specific subsystem. If all subsystems on a bus stops responding, a bus error flag is set. Once a transaction is successful, the error flags are cleared and recovery procedures are only activated on the second failure.

I²C error (ADCS or COMM):
The OBC will attempt an I²C restart on the specific system on the main bus. If this fails, the OBC will ask the EPS to power-toggle that system.

I²C error (EPS):
An I²C restart attempt will be made on both the main and backup bus. If this still fails, the EPS is set as “not responding” and communication towards the EPS is disabled.

I²C error (Payload):
The OBC will ask the EPS to power-toggle the Payload.

Bus error:
The OBC will attempt to make the EPS power-toggle all subsystems (including the OBC) and the I²C request will be made on both the main and backup bus. If the bus error continues, a bus failure flag is set.

Bus failure:
Once the system gets a bus failure, all future communication will be attempted on both busses.

Unexpected return values:
If any I²C transaction returns unexpected values, a CRC check will be run where available, an error will be logged and the system in question will be power-toggled.

6.5. MEMORY

Because the data storage memory does not experience bit errors, no error correcting code has been implemented in the OBC software. The hamming code is implemented for error prevention, while the CRC bytes in the diagnostic data are implemented to limit data loss in the downlink process.

The memory is grouped into two programming structs, an I²C driver struct and a complete OBC data struct. Using the EBI interface on the XMEGA allows the internal data bus to access the external memory space seamlessly. This allows the software to read and write directly to the struct variables, just as it would with variables in the internal memory.

Due to the GCC compiler, memory is limited to a 64 KB address space. The first 0x4000 (hex) are not available due to being reserved for the XMEGAs

own memory locations. This means the external memory address space will range from 0x4000 to 0xFFFF.

The I^2C struct is located at 0x4000 and the OBC struct is located after the I^2C struct at 0x4050.

The OBC data struct contains:

- All global OBC flags, counters and variables.
- 127 diagnostic/beacon packages, with index variables.
- 1024 events, with index variables.

127 diagnostic packages will be enough for 10 hours and 35 minutes of normal operation. This meets the requirements found in “4.5 ORBIT AND BUDGET ANALYSIS”. The 1024 events is the equivalent of 64 data packages. It would take about 2 hours and 50 minutes to fill the event buffer if a new event occurred every 10 seconds. This should be enough, as it is not expected to be more than a few dozen events during startup and hopefully none during normal operation.

The index variables work as a circular buffer, with a fill counter. This allows the entire memory space to be used and maintains a variable that tells the ground crew how many packages are available.

It is not yet decided if the buffer will overwrite old data or not, however this is a minor change in the case-logic of the software.

6.6. RTC CLOCK

The main functionality for the RTC Clock is to provide a wake-up tick and time control for when tasks are performed in the satellite. During the RTC interrupt, the satellite time is updated and beacon, status and diagnostic collection flags are activated at their given intervals.

Status collection occurs at every tick (10 seconds), beacon every three ticks (30 seconds) and diagnostic collection every 30 ticks (5 minutes). The beacon and diagnostic intervals can however be changed from the ground station if desired.

Satellite time is given in a 5 byte format that includes an 8 bit week counter, a 16 bit tick counter (10 seconds/tick) and a 16 bit RTC counter (1/40960 tick).

This means the maximum time resolution of the satellite is 1/4096 second, or ~244 µs, which is more accurate than you can get using either NTP, or a single long-wave radio time source.

The accuracy and drift estimated in “5.4 CRYSTAL OSCILLATOR” shows we cannot expect any better than ~500 ms difference due to frequency tolerance and ~50 ms temperature compensated drift per day. The drift requires an update interval of max 7 hours in order to keep the internal satellite clock within 15 ms of International Atomic Time (IAT), not including any constant offset and assuming a predictable synchronization with IAT. This is by all intents and purposes under ideal circumstances. Further testing is required to gain accurate numbers.

6.6.1. Synchronization

The method of synchronization follows the same concept as the Network Time Protocol [42]. This means calculating the round-trip delay time and an offset value.

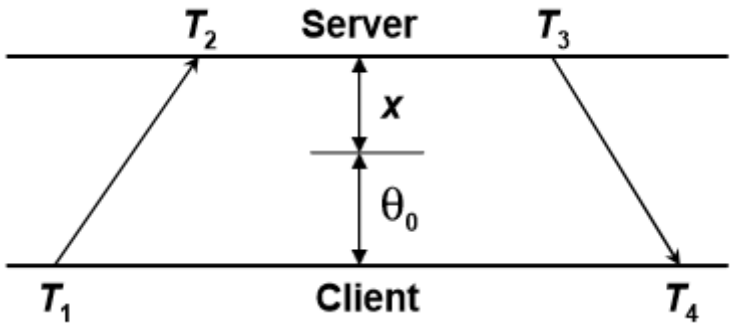


Figure 6-9 - NTP Synchronization

In order to find the round-trip delay time and offset value, four raw timestamps are required as shown in “Figure 6-9”. The client original timestamp T_1, the timestamp for message received by the server T_2, a new timestamp for when the server sends a reply T_3 and the client timestamp for receiving the reply T_4.

The clock offset is found with the following formula:

$$\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

The round-trip delay is found using this formula:

$$\delta = (T_4 - T_1) - (T_3 - T_2)$$

The clients new time will be $T_4 + \theta$ and the maximum difference between the client and server time will be $\delta/2$. For the client and server to get identical clocks, the time to transfer the message in both directions needs to be identical.

In the implementation on the satellite, the OBC is the client and T_1 is the beacon signal. The server would be the ground station, sending a "time update" command. Depending on distance, package length and decoding time, the ground station can compensate the return value to make the propagation delay appear the same in both directions. In order for the satellite's on-board clock to give any meaning, the ground station needs an accurate reference clock compared to IAT. This can be achieved by using a NTP service (good), a system with a GPS receiver or atomic clock (best).

6.7. MISCELLANEOUS

6.7.1. Software Power Reduction

To reduce static power consumption on the OBC hardware, the software has included several methods to reduce the XMEGAs power drain:

- All unused modules are powered off with a special power reduction register in the XMEGA.
- Unused pins have an internal pull-up to VCC, in order to avoid internal switching and leakage.
- The EEPROM, watchdog and JTAG is disabled.
- The brown out detector is put in sample mode.

These methods were recommended in the "AVR1010" application note.

6.7.2. CubeSTAR.h

In order to make the integration between the COMM, EPS and OBC subsystems, a CubeSTAR.h header file was created that included all I2C bus protocol definitions, commands, transfer lengths and similar. It was not fully adopted by the other subsystems, but is used on the OBC software and provides a useful reference for future integration work.

6.7.3. Boot Loader

A full boot loader was developed for the XMEGA, which allowed both I^2C master and slave operation. This would let an external I^2C device upload new firmware to a device, or the device itself could collect the firmware from an external source and reprogram itself.

The benefit of having a boot loader on the microcontroller would be the ability to reprogram the firmware in case of a radiation bit flip. In case programming mistakes were fixed, or new features were implemented, they could be uploaded from the ground station and the new software updates could be reprogrammed by the boot loader.

The boot loader has not been included in the system, as it would introduce more hardware complexity (i.e. external I^2C memory). Using the boot loader would introduce more vulnerabilities in the satellite. E.g., something could go wrong in the reprogramming process. This was a joint decision made between the COMM, EPS and OBC system. The radiation environment was not considered serious enough to cause a problem for flash memory. Since the firmware is stored on flash memory, the potential dangers of firmware error due to a SEU is considered negligible.

6.7.4. Watchdog

A watchdog was not implemented on the OBC module. This was because the time constraints in the external and internal watchdogs did not match the OBC sleep cycle. Instead of having the watchdog feature on the OBC module, the feature was implemented on the EPS controller. If the OBC stops sending periodic status checks to the EPS, the EPS subsystem will power-toggle the OBC.

6.7.5. SD Card with FAT file system

Early in the project, the OBC was in charge of sensor monitoring and logging of data to a SD card. By using a FAT file system, the SD card could be put directly into a computer and the log file could be read.

This concept was discarded, as the implementation of both the SD card driver and FAT file system was extensive and slow. It required 32-bit variables and significant amount of data shuffling. As it would not be useful for launch, further development was cancelled.

Chapter 7:

Tests & Results

7.1. HARDWARE

7.1.1. Power Consumption

Static power consumption has been measured on two versions of the hardware. The measurements were made while the processor is in sleep mode so only static power consumption is measured. The initial design, version 0.2.3, had the following characteristics:

- 15.8 mA without RC oscillator running.
- 17.2 mA with internal 2 MHz RC oscillator running.
- 27.3 mA with internal 32 MHz RC oscillator running.

After the initial design, the power regulator and power switch was added to the design and the layout was simplified by removing various components that were not used. Power consumption on this hardware, version 1.1.0, was only measured while power to memory was disabled. This showed a static consumption of ~2 mA.

Active power consumption can peak at 100 to 150 mA, depending on how much memory read and write operations are performed.

As the sleep cycle is a full 10 seconds, the total power consumption will be dominated by the static power drain. Duty cycle for the software has been difficult to measure, as all the subsystems have not yet been completed. It is however estimated to be a few milliseconds, except during extensive operations like sending telemetry. The significant reduction in static power consumption means the OBC hardware will require very little power.

7.1.2. Crystal Tolerance & Stability

Several tests were performed with different reference sources. The XMEGA has two internal RTC clock sources, one is the internal 32768 Hz RC oscillator and the other is a 1024 Hz ULP oscillator.

By writing the RTC value to a PC at regular intervals, the drift in the oscillators could be found. The test showed an accuracy of ~1-2% on the internal 32768 Hz RC oscillator and the 1024 Hz ULP had ~10% accuracy. The thermal characteristics of both oscillators would be very bad. This confirms the characteristics given by ATMEL in their "AVR1003" application note, where they estimate the ULP at 30% accuracy and the 32768 Hz internal RC oscillator at 1%. An external clock source is needed to achieve the desired accuracy.

Two factors are required in order to test the external 32768 Hz crystal tolerance and stability. Based on the curve shown in "Figure 5-4 - Crystal Frequency by Temperature", two unknown variables need to be found. T_0 will define the nominal temperature where the crystal has its nominal frequency. This is defined to be between 20 and 30 degrees Celsius. Then it is required to find F_0, the nominal frequency at the nominal temperature. It is defined as 32768 Hz $\pm 10\ ppm$ not including other factors like crystal aging and capacitive load.

Because the two unknown variables will vary with each crystal, this test must be performed on the flight model of the hardware, if the desired accuracy is to be achieved. As the temperature coefficient and the capacitance ratio of the crystal is unknown, the temperature drift should be measured as well.

The calculations done in "5.4 CRYSTAL OSCILLATOR" show the margins of error are very small if you want to achieve the desired accuracy of max 15 ms. The only way to achieve an accuracy close to 15 ms is by measurement of the final flight model hardware and a good synchronization method.

7.2. SOFTWARE

Testing of software was done in individual blocks and system wide.

7.2.1. Memory

The EBI and memory was tested with a memory-mapping test, to make sure both the hardware connection and the software memory map was working as expected. The intention was to verify that the entire memory could be written and read in the correct order. A series of seven numbers were selected and then written to memory in a loop until the memory was full. The values was then verified to be the correct number by reading the memory in the same order as they were written. This would show if there was an error in the memory mapping or connection error causing a memory overwrite. In order for this to work, the amount of numbers had to be equal to a prime number.

All the values were then inverted and the test was run again to see if there were any stuck bits in the memory. An additional test where the memory was filled with only 0x00 and 0xFF would verify all bits could be 1 and 0.

7.2.2. Peripheral Configurations

All peripheral and hardware configurations in the XMEGA was verified to be correct. This included the system clock frequency with automatic temperature calibration, the interrupt sources, I^2C communication and GPIO ports.

7.2.3. I^2C LabVIEW Application

The I^2C LabVIEW application was made to provide subsystems the ability to test the internal bus protocol and communication, without connecting it to the OBC module.

The LabVIEW application could be run on any computer with LabVIEW and the I^2C LabVIEW module driver installed. The external interface to any subsystem was only three wires, a 2-pin I^2C connector and a ground reference.

This application could only run as I^2C master and could not work as a slave. This was due to limitations in the LabVIEW module hardware and the interface to the LabVIEW software running on the PC.

All students tested their subsystems with this device before doing integration work together with the OBC. This allowed the software logic to be tested completely before introducing more sources of errors.

“Figure 7-10” shows the LabVIEW application interface where the user can type the transaction details, and display the result.

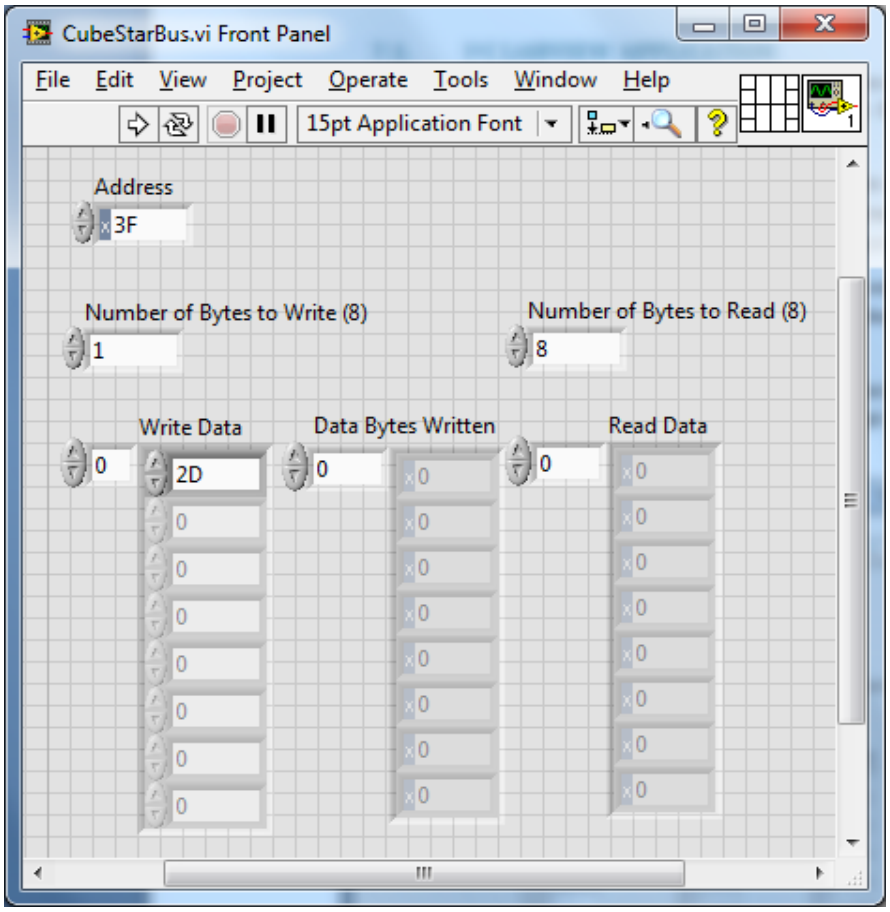


Figure 7-10 - LabVIEW Application Interface

7.2.4. PC Application / Simulation

The PC Application was developed to be the main debugging tool for the OBC software. All debugging text from the OBC was displayed in this program and it allowed verification that the software was running as intended.

A feature to log all collected sensor data and copy them into excel was implemented. This allowed beacon/diagnostic packages to be sent internally in the satellite and then logged to a text file using the PC interface. The data could then be copied into an excel spreadsheet, where all the cells

automatically calculated and formatted the values to the user. An example of how the EPS diagnostic data is presented is show in “Figure 7-11”.

Tests were performed between COMM, EPS and OBC and the system was operating as expected.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1					sOn										vBat1	max	min	vBat2			vBat3			vBat4		
2	2D	0E	0	0	55	CC	0	0	2D	FF	FF	0	0	0	3310	3315	3310	3311	3316	3311	3313	3316	3311	3311	3316	3311
3	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3311	3311	3311	3313	3313	40959	3311	3313	3311	3311	3311	3311
4	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3311	3311	3311	3311	3311	3311	3313	3313	40959	3311	3311	3311
5	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3311	3311	3311	3311	3311	3311	3311	3313	3311	3311	3311	3311
6	2D	0E	0	0	55	CC	0	0	2D	FF	FF	0	0	0	3310	3315	3310	3311	3316	3311	3311	3316	3311	3311	3316	3310
7	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3313	3311	3311	3313	3311	3311	3311	3310
8	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3313	3311	3311	3313	3311	3311	3311	3310
9	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3310	3311	3310
10	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3311	3311	3310
11	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3311	3311	3310
12	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3310	3311	3310
13	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3311	3311	3310
14	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3311	3311	3310
15	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3311	3311	3310	3311	3311	3311	3311	3313	3311	3311	3311	3310
16	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3313	3311	3310	3311	3310
17	2D	0E	0	0	55	0	0	0	0	FF	FF	0	0	0	3310	3311	3310	3311	3311	3311	3311	3311	3311	3311	3311	3310

RawData | Decimal | Converted

Figure 7-11 - Excel w/ EPS Diagnostic Data

7.2.5. OBC Test Module

The OBC test module was made as a companion to the PC Application and made it possible to send any I^2C command to any module just like the I^2C LabVIEW Application. Using the OBC module hardware, it could simultaneously run the OBC software and perform a more realistic test setup together with the other subsystems. This allowed the OBC to work as an I^2C slave and it added the possibility of testing the I^2C backup bus in case of main bus failure.

Tests were performed with the EPS and COMM subsystems using the satellite backplane and all commands were tested successfully. The ADCS subsystem did a separate test using this module and the I^2C communication was working as expected. A full integration test with the ADCS or payload subsystems has not been done prior to the completion of this thesis.

Chapter 8:

Discussion & Conclusion

The work on this thesis has been the design and implementation of an OBC system for the CubeSTAR satellite. During the design and implementation, the following features have been completed:

- A complete hardware module (Ver. 1.1.0) for the OBC system has been developed, implemented and tested. The hardware includes an ATXMEGAA1U microcontroller with a temperature predictable crystal and a radiation bit-flip immune memory.
- A full I^2C bus protocol for communication between subsystems in the CubeSTAR satellite and the ground station.
- OBC software that runs autonomous and collects diagnostic data. It will monitor satellite operation and attempt recovery on unresponsive subsystems.
- 2 mA idle power consumption, with duty cycles down to a few milliseconds every ten seconds.
- Two separate debug systems for verification of the communication protocol and subsystem integration.

The OBC has a maximum of 10 seconds response time to any problems that might occur in the satellite. It will collect information from subsystems on demand and maintain historical data up to 10-11 hours. A beacon signal is implemented to notify any ground station of the satellite's presence.

Based on the standard COTS TID characteristics, the OBC module should remain operational for at least 3 to 4 months. This would depend on the TID rating of the microcontroller and the amount of TID gained while in orbit.

Both factors are difficult to predict without further analysis, but the typical COTS characteristics satisfies the mission objective that states a minimum of 2 months.

The issues with SEUs for the OBC are handled by using a bit-flip immune memory and hamming code. If a component on CubeSTAR experiences a SEL, each subsystem has overcurrent protection in hardware that should power off the device. The SEL induced current may not be high enough to trigger the overcurrent protection. If this is the case, the EPS will power-toggle the subsystem on command from the OBC or ground station.

The clock precision and accuracy might not be satisfactory for the payload scientific measurements. It is estimated the internal clock will have at least 500 ms tolerance drift and 50 ms temperature compensated drift per day. There are two alternatives to improve the clock accuracy. Calibration and measurements on the flight model hardware, with post processing, or use a GPS time reference. GPS is under International Traffic in Arms Regulations (ITAR) and it is uncertain whether this would be possible to implement on the satellite or not.

Subsystem integration has been completed without significant complications. The communication protocol allows subsystems to get communication up and running without implementing much logic on top of the basic I^2C driver.

8.1. PROBLEMS ENCOUNTERED

8.1.1. I^2C Communication

During integration, the I^2C bus would become stuck in a loop of interrupts that would stop the program from running. This problem would come and go at random intervals, with little or no change to the source code. This issue disappeared at some point during the development.

It is assumed this was caused by an error in the test setup, but this error is included in this thesis in case it should reappear later in the integration work. In such a case, further analysis of the I^2C communication should be performed.

8.1.2. AVR GCC Compiler

Late in the project, it was discovered that the AVR GCC compiler only had support for 16-bit pointers. This meant the address space was limited to 0xFFFF (hex), or 64 KB of memory locations. There were two alternatives for reading and writing to memory locations beyond the initial 16-bit pointers. Either custom assembly functions could be implemented or the software could be rewritten for a commercial compiler like IAR or CodeVision. Both methods would unfortunately require a lot of work and time.

In addition, the internal pointers in the XMEGA were only 16 bit and they used special RAMP registers that allowed an address space up to 24 bit. Writing to these would create an additional overhead on transaction time when reading and writing from memory addresses above 0xFFFF. It was decided to keep the memory at 64 KB, as it was significant enough for the minimum requirements. The additional time required to implement 24-bit pointers could not be justified.

8.2. FUTURE WORK

8.2.1. Radiation Analysis

CubeSTAR will travel in a radiation environment where the electronics can experience both SEU and SEL. As the environment, error rate and characteristics of the subsystems on board CubeSTAR is unknown, the satellite will risk failing early in its mission due to radiation effects.

A thorough analysis of the radiation environment for CubeSTAR's orbit is recommended. The TID rating and SEL threshold should be found on critical components.

8.2.2. Integration Work

A complete satellite integration test is required to verify the satellite can run autonomous for long periods. It will also be needed to complete the OBC software, as some features are dependent on all subsystems being available and responsive. During the complete integration testing, statistical power consumption by the OBC can be measured and logged.

Further integration work towards the ADCS and Payload is needed. Once those subsystems have progressed further in development, the software

features that integrate these subsystems with the OBC needs to be completed.

In order to have an accurate post processing of CubeSTAR's timestamps, a measurement of drift and tolerance in the RTC on the flight model hardware is required. Calculations and tests for synchronization between the ground station and the OBC must be performed. Transfer time, round trip and accuracy of synchronization is required. The method for synchronization must be implemented in the ground station.

8.2.3. Hardware Ver. 1.2.0

After all the hardware was completed and implemented, a new generation of XMEGA microcontrollers were found as they had recently been released. This changed the hardware design for CubeSTAR from the first generation microcontroller ATXMEGAA1 to the second generation ATXMEGAA1U. As the two devices were pin-swap compatible, the processor was changed and implemented in version 1.1.0 of the hardware.

The ATXMEGAA1U had some additional benefits over the old ATXMEGAA1, which allowed the EBI memory interface to be implemented without external latch components. A new schematic and layout was completed with optimizations for the ATXMEGAA1U (Ver. 1.2.0). The PCB was produced, but the hardware has not been implemented or tested. Software is still compatible with both versions of the board and there are no other relevant changes. Completing the new board is recommended as it reduces potential sources for errors and reduces power consumption slightly.

8.2.4. Work Related to Memory

Implementing memory access beyond the first 16-bit address field would increase storage space for the satellite and reduce the potential for information to be lost. This would improve the functionality of the satellite.

Having 24-bit memory addresses would enable the OBC to use the internal memory for boot loading. A new boot loader would need to be written, but this is a potential improvement to the OBC redundancy.

If this issue is not addressed in software, the memory device should be changed from the MR2A08A to the MR0A08B. If this device is built on the 130 nm process, it is possible it will have better SEL characteristics.

Index

G

I

J

L

M

T

U

V

References

[1] J. Moen, "ICI-1: A NEW SOUNDING ROCKET CONCEPT TO OBSERVE MICRO-SCALE PHYSICS IN THE CUSP IONOSPHERE," University of Oslo, Oslo.

[2] CalTech, "The CubeSat: The Picosatellite Standard for Research and Education," Aerospace Engineering Department California Polytechnic State University, San Luis Obispo, CA.

[3] T. A. Bekkeng, "Prototype Development of a Multi-Needle Langmuir Probe System," University of Oslo, Oslo, 2009.

[4] J. L. Tresvig, "Design of a Prototype Communication System for the CubeSTAR Nano-satellite," University of Oslo, Oslo, 2010.

[5] HAARP, "Total Electron Content," [Online]. Available: http://www.haarp.alaska.edu/cgi-bin/ashtech/tec.cgi. [Accessed 30 January 2013].

[6] M. H. Ilknur Baylakoglu, "Reliability Concerns of Radiation Effects on Space Electronics," Space Technologies Research Institute, Ankara, Turkey.

[7] M. Poizat, "Space Environment and Effects," in *Space Radiation and its Effects on EEE Components*, EPFL Space Center, 2009.

[8] K. E. Holbert, "Single Event Effects," 18 January 2006. [Online]. Available: http://holbert.faculty.asu.edu/eee560/see.html. [Accessed 28 January 2013].

[9] A. Tylka, "Single Event Upsets Caused By Solar Energetic Heavy Ions," *Nuclear Science, IEEE Transactions on,* vol. 43, no. 6, pp. 2758 - 2766, 1996.

[10] "Single Event Upsets (SEUs)," [Online]. Available: http://www.spaceweather.go.kr/effect/english/07_03_03. [Accessed 28 January 2013].

[11] P. Stauning, "High-Energy Particle Radiation Effects in the Instruments and Memory Circuits of Low-altitude Satellites.," ESA, ESTEC, Ørsted Satellite, 2000.

[12] F.Sturesson, "Single Event Effects (SEE) Mechanism and Effects," in *Space Radiation and its Effects on EEE Components*, EPFL Space Center, 2009.

[13] NASA, "SPACE RADIATION EFFECTS ON ELECTRONIC COMPONENTS IN LOW-EARTH ORBIT," 1996.

[14] J. M. Benedetto, "Economy-Class Ion-Defying ICs In Orbit," *Spectrum, IEEE,* vol. 35, no. 3, pp. 36 - 41, 1998.

[15] Maxwell Technologies, "How Rad Hard Do You Need? The Changing Approach To Space Parts Selection?".

[16] P. Dodd, "Current and Future Challenges in Radiation Effects on CMOS Electronics," *Nuclear Science, IEEE Transactions on,* vol. 57, no. 4, pp. 1747 - 1763, 2010.

[17] P. P. Shirvani, "Software-Implemented EDAC Protection Against SEUs," *Reliability, IEEE Transactions on,* vol. 49, no. 3, pp. 273 - 284, 2000.

[18] "History Of Hamming Codes," [Online]. Available: http://biobio.loc.edu/chu/web/Courses/Cosi460/hamming_codes.htm . [Accessed 12 April 2012].

[19] C. B. S. IIsie-Chia Cliarig, "A (208,192;8) Reed-Solomon decoder for DVD application," in *Communications, 1998 IEEE International Conference on*, Hsinchu, Taiwan, 1998.

[20] CubeSat Cal Poly, "CubeSat Design Specification Rev. 12," 2009. [Online]. Available: http://www.cubesat.org/images/developers/cds_rev12.pdf.

[21] D. M. A. E. M. O. N. M. Omid Zeynali, "The Design and Simulation of the Shield Reduce Ionizing Radiation Effects on Electronic Circuits in Satellites," 2011.

[22] Spenvis, "Background: Trapped particle radiation models," [Online]. Available: http://www.spenvis.oma.be/help/background/traprad/traprad.html.

[23] C. Poivey, "In-Flight Observations of Long-Term Single- Event Effect (SEE) Performance on Orbview-2 Solid State Recorders (SSR)," in *Radiation Effects Data Workshop, 2003 IEEE*, Greenbelt, USA, 2003.

[24] M. A. Grønstad, "Implementation of a communication protocol for CubeSTAR," University of Oslo, 2010.

[25] M. Oredsson, "Electrical Power System for the CubeSTAR Nanosatellite," University of Oslo, 2010.

[26] K. Rensel, "An Attitude Detumbling System for the CubeSTAR Nano Satellite," University of Oslo, 2011.

[27] B. Hallgren, "SEE and TID Qualification of the ELMB128 Series Production," ATLAS Internal Working Note, Geneva, Switzerland, 2004.

[28] O. Mancini, "Tutorial Precision Frequency Generation Utilizing OCXO and Rubidium Atomic Standards with Applications for Commercial, Space, Military, and Challenging Environments," 18 March 2004. [Online]. Available: http://www.ieee.li/pdf/viewgraphs/precision_frequency_generation.pdf.

[29] Crystek Crystals Corp., "Pierce-gate oscillator crystal load calculation," 2004.

[30] STATEK Corporation, "The quartz crystal model and its frequencies," Orange, CA, USA.

[31] Atmel, "ATXMEGA IBIS Files," [Online]. Available: http://www.atmel.com/images/ibisxmega.zip.

[32] SEAKR Engineering, Inc., "Memory Technology for Space," 2009.

[33] S. G. a. D. N. Leif Scheick, "SEU Evaluation of FeRAM Memories for Space Applications.," Jet Propulsion Laboratory, Pasadena CA, USA.

[34] C. S. a. M. Tranchero, "Use of FRAM Memories in Spacecrafts (p.217)," InTech - ISBN: 978-953-307-456-6, 2011.

[35] D. N. Nguyen, "TID Testing of Ferroelectric Nonvolatile RAM," in *Radiation Effects Data Workshop, 2001 IEEE*, Pasadena CA, USA, 2001.

[36] J. Heidecker, "MRAM Technology and Status," in *NEPP Electronic Technology Workshop*, Greenbelt MD, USA, 2012.

[37] Aeroflex, "Aeroflex16Mb / 64Mb MRAM Development Status," 2011.

[38] J. Heidecker, "Single Event Latchup (SEL) and Total Ionizing Dose (TID) of a 1 Mbit Magnetoresistive Random Access Memory (MRAM)," Radiation Effects Data Workshop, 2010 IEEE, Pasadena CA, USA, 2010.

[39] J. L. L. S. T. M. S. S. a. D. M. Romney R. Katti, "Heavy-Ion and Total Ionizing Dose (TID) Performance of a 1 Mbit Magnetoresistive Random Access Memory (MRAM)," in *Radiation Effects Data Workshop, 2009 IEEE*, Plymouth MN, USA, 2009.

[40] A. McCalden, "Electronic Component Specification - Document Number: MSSL/SLB-EIS/SP020.02," MULLARD SPACE SCIENCE LABORATORY, London, UK, 2010.

[41] NASA, "Radiation Effects Assessment of MRAM Devices," JPL Publication, Pasadena CA, USA, 2008.

[42] NTP, "How NTP Works," NTP Standard, 18 August 2012. [Online]. Available: http://www.eecis.udel.edu/~mills/ntp/html/warp.html. [Accessed 31 January 2013].

[43] T. Tsugawa, "Ionospheric disturbances detected by GPS total electron content observation after the 2011 off the Pacific coast of Tohoku Earthquake," *Earth Planets and Space,* vol. 63, no. 7, p. 875–879, 2011.

Table of Figures

List of Tables

Appendix A: I2C Bus Protocol

This CubeSTAR TECHDOC will provide documentation and explanation of the internal bus commands of the CubeSTAR satellite.

The current revision is;
“CubeSTAR-TECHDOC-Main Bus Protocol v12”

Always refer to the most recent version of the document.

A.1. Intended Audience

Anyone involved with the CubeSTAR project that needs insight into the satellites main bus commands.

A.2. Purpose

By reading this document, you will know how to implement a module that responds to the commands, and how to debug the transactions on the main I2C bus.

A.3. Prerequisites

The reader of this document should:

- Be familiar with the I2C protocol.
- Have technical insight in some part of the CubeSTAR project.

A.4. I²C ADDRESS LIST

This table shows all device addresses on the main I²C bus:

DES	HEX	TYPE	DESCRIPTION
10	0A	PAYLOAD	Accessed by OBC
11	0B		
12	0C	COMM	Accessed by OBC
13	0D		
14	0E	EPS	Accessed by OBC
15	0F		
16	10	ADCS	Accessed by OBC
17	11		
18	12	OBC	Accessed by COMM
19	13		

A.5. HAMMING VALUES

Hamming encoded values are often used for flag bytes on the I²C bus. The conversion between a hamming and hex value can be seen in this table:

Hamming	Hex Value
0	0x00
1	0x1E
2	0x2D
3	0x33
4	0x4B
5	0x55
6	0x66
7	0x78
8	0x87
9	0x99
A	0xAA
B	0xB4
C	0xCC
D	0xD2
E	0xE1
F	0xFF

A.6. ADCS – ATTITUDE DETERMINATION & CONTROL SYSTEM

Command Name	Command	Response
ADCS Status	0x1E	1 byte
ADCS Diagnostic	0x2D	62 bytes
ADCS Restart	0x55	None
ADCS Ground Station Command	0x66 + 256 bytes	256 bytes

A.6.1. ADCS STATUS

The ADCS Status response is a byte that includes four flags.

- 8 - ADCS_USE <Unhandled System Error/Warning>
- 4 - ADCS_NSE <New System Event>
- 2 - ADCS_ANC <Attitude Not Correct>
- 1 - ADCS_OFF <ADCS Offline>

ADCS_USE equals **true** when any unexpected software/firmware error has occurred. It can be true if the ADCS has detected a hardware problem it cannot correct, like firmware data corruption, or the attitude has gone out of the required range. This flag will tell the OBC it needs to investigate and possibly attempt a correction, or intervention.

ADCS_NSE equals **true** when any event worth notifying about has occurred in the ADCS. This would include events such as ADCS reboots, the attitude has reached its target, and similar notifications. This flag will tell the OBC something has happened, so it can gather event data and store it.

ADCS_ANC equals **true** when the satellite attitude and spin rate is not within accepted tolerances for the payload to start sampling.

ADCS_OFF equals **true** if the ADCS is in idle mode.

Flag	Value	Mask	Position
ADCS_USE	8	(1<<3)	3
ADCS_NSE	4	(1<<2)	2
ADCS_ANC	2	(1<<1)	1
ADCS_OFF	1	(1<<0)	0

Transaction Details:

Command length: 1 byte
Response length: 1 byte

Command byte value: 0x1E
Response byte value: Any of the 16 hamming values.

The response value is the hamming value found by adding the flag values together. If a flag is not implemented, or false, the value for that flag is 0.

Value Example:
The ADCS_NSE and ADCS_ANC flags are true, and the other flags are false, the hamming value to send as response would be 4+2 = Hamming 6.
Hamming 6 = 0x66.

A.6.2. ADCS DIAGNOSTIC

The ADCS Diagnostic response consists of several values. For dynamic variables (sensors), the new value must be more recent than the last time an ADCS Diagnostic request was made, or no longer than 5 minutes old.

- Package ID:
  - This is a fixed value of 0x2D, which is the same as the command byte for “ADCS Diagnostic”.
- ADCS ID:
  - The pre-defined ID for ADCS, which is the same as its I2C slave address.
- Event Data:
  - Event data to send when using the ADCS_NSE flag.
- Sensor Values:
  - Sensor data from the ADCS. Min, Max and Average values can be used where appropriate. If no values exist, write them to 0x00.

Byte #	Name	Value
1	Package ID	0x2D
2	ADCS ID	0x10
3-13	Event Data	Arbitrary
14-62++	Sensor values	Actual

Transaction Details:

Command length: 1 byte
Response length: 62 bytes

Command byte value: 0x2D
Response byte values: 62 bytes as described.

Value Example:

No feature is implemented, and no sensor values have been collected yet.

Byte #	1	2	3	4	5	6	7	8	9	10	11+
Value	0x2D	0x10	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

A.6.3. ADCS RESTART

The ADCS system will restart immediately upon receiving this command.

Transaction Details:

Command length: 1 byte
Response length: 0 bytes

Command byte value: 0x55
Response byte values: Not Available

Value Example:

Not Available. No values to return.

A.6.4. ADCS GROUND STATION COMMAND

The UiO Ground Station will send command data to the satellite. These commands will be included in this protocol when available.

Transaction Details:

Command length: 3-257 bytes
Response length: 0-256 bytes

Command byte value: 0x66
Response byte values: Up to 256 bytes (Optional)

GSC: ADCS Activate Magnetic Coils

This makes the ADCS turn on specific magnetic coils, and should put the ADCS into idle mode automatically.

Transaction Details:

Command length: 2 bytes
Response length: 0 bytes

Command byte value: 0x87
Response byte values: Not Available

The first byte is the command byte 0x87, telling the ADCS this is an ACTIVATE MAGNETIC COILS command.

Second byte tells the ADCS which magnetic coils should be turned on, and is hamming coded.

Flag	Value	Mask	Position
Coil 1	8	(1<<3)	3
Coil 2	4	(1<<2)	2
Coil 3	2	(1<<1)	1
Coil 4	1	(1<<0)	0

Value Example:

These 2 bytes are sent to the ADCS:

Byte 1	Byte 2
0xAA	0x33

This means Coil 1 and 2 should be turned on. The return value should then be the same as byte 2: 0x33.

A.7. COMM – COMMUNICATION

Command Name	Command	Response
COMM Diagnostic	0x2D	13 bytes
COMM Application CRC	0x4B	24 bit CRC
COMM Restart	0x55	None
COMM Ground Station Command	0x66 + 256 bytes	256 bytes
COMM Send Data (end transfer)	0xAA + 256 bytes	0x55
COMM Send Data (continue)	0xB4 + 256 bytes	0x4B
COMM Antenna Deploy	0xE1	See details

A.7.1. COMM DIAGNOSTIC

The COMM Diagnostic response consists of several values. For dynamic variables (sensors), the new value must be more recent than the last time a COMM Diagnostic request was made, or no longer than 5 minutes old.

- Package ID:
  - This is a fixed value of 0x2D, which is the same as the command byte for “COMM Diagnostic”.
- COMM ID:
  - The pre-defined ID for COMM, which is the same as its I2C slave address.

Byte #	Name	Value
1	Package ID	0x2D
2	COMM ID	0x0C
3-13++	Misc values	Arbitrary

Transaction Details:

Command length: 1 byte
Response length: 13 bytes

Command byte value: 0x2D
Response byte values: 13 bytes as described.

Code Example:

```
uint_8t response_array[9] = {0x2D, 0x0C, 0x00…  0x00, 0x00};
// copy values into array
// send response_array[I2C_Bytes_Sent] on I2C read request.
```

A.7.2. COMM APPLICATION CRC

This command tells the COMM system to return the application CRC values. If there are not any CRC values available, the COMM system will return bytes of 0x00 only. Once the transaction is completed, a new CRC value must be generated. If the new CRC is incorrect, the COMM_USE flag should be set.

Transaction Details:

Command length: 1 byte
Response length: 3 bytes

Command byte value: 0x4B
Response byte value: 24 bit CRC

Value Example:

No CRC values have been generated yet.

Byte #	1	2	3
Value	0x00	0x00	0x00

Code Example:

```
uint_8t response_array[3] = {0x00, 0x00, 0x00};
// copy CRC values into array
// send response_array[I2C_Bytes_Sent] on I2C read request.
```

A.7.3. COMM RESTART

The COMM system will restart immediately upon receiving this command.

Transaction Details:

Command length: 1 byte
Response length: 0 bytes

Command byte value: 0x55
Response byte values: Not Available

Value Example:

Not Available. No values to return.

Code Example:

```
if(0x55) //if reset command
{
  ccp_write_io( &RST.CTRL, RST_SWRST_bm );
  delay_us(1000); //in case of reset failure, no deadlock
}

… //external assembly code:

  .global ccp_write_io
  .section .text.ccp_write_io, "ax", @progbits
  .type ccp_write_io, @function
  ccp_write_io :

  out     RAMPZ, r1              // Reset bits 23:16 of Z
  movw    r30, r24               // Load addr into Z
  ldi     r18, CCP_IOREG         // Load magic CCP value
  out     CCP, r18               // Start CCP handshake
  st      Z, r22                 // Write value to I/O register
  ret                            // Return to caller

  .size   ccp_write_io, . - ccp_write_io
```

A.7.4. COMM GROUND STATION COMMAND

The UiO Ground Station will send command data to the satellite. These commands will be included in this protocol when available.

Transaction Details:

Command length:	3-257 bytes
Response length:	0-256 bytes
Command byte value:	0x66
Response byte values:	Up to 256 bytes (Optional)

A.7.5. COMM SEND DATA & END TRANSACTION

This is a telemetry command that will contain the first command byte followed by a 256 byte data package. After this package is sent to the ground station, the transaction should finish.

Transaction Details:

Command length:	257 bytes
Response length:	1 byte
Command byte value:	0xAA
Response byte value:	0x55

A.7.6. COMM SEND DATA & CONTINUE TRANSACTION

This is a telemetry command that will contain the first command byte followed by a 256 byte data package. After this package is sent to the ground station, the antenna should preamble while waiting for the next package.

Transaction Details:

Command length:	257 bytes
Response length:	1 byte
Command byte value:	0xB4
Response byte value:	0x4B

A.7.7. COMM ANTENNA DEPLOY

Upon receiving this command, the COMM system will deploy the antenna. The COMM system will respond with the antenna status. If this feature is not implemented, the COMM system will respond with 0x00.

Transaction Details:

Command length: 1 byte
Response length: 1 byte

Command byte value: 0xE1
Response byte values: 0xE1, 0x1E, or 0x66

Value Example:

If the antenna is not deployed, the following values are valid:

- Antenna deploying: 0x1E
- Deployment failed: 0xE1

If the antenna is deployed, the following values are valid:

- Antenna is deployed: 0x66
- Antenna error: 0xE1

Code Example:

```
If(0xE1) //if antenna deploy
{
  If(antenna_not_deployed)
  {
    //deploy antenna
    //if success, set status to deployed, else set to error
  }
  //return antenna/command status
}
```

A.8. EPS – ELECTRONIC POWER SYSTEM

Command Name	Command	Response
EPS Status	0x1E	1 byte
EPS Diagnostic	0x2D	161 bytes
EPS Application CRC	0x4B	24 bit CRC
EPS Restart	0x55	None
EPS Ground Station Command	0x66 + 256 bytes	256 bytes
EPS Power Subsystem	0x87 + 2 bytes	1 byte
EPS BW	0x99 + 4 bytes	None
EPS TCS	0xAA + 3 bytes	None
EPS Uptime	0xD2	3 bytes
EPS m-NLP Deploy	0xE1	See details

A.8.1. EPS STATUS

The EPS Status response is a byte that includes four flags.

- 8 - EPS_USE <Unhandled System Error/Warning>
- 4 - EPS_NSE <New System Event>
- 2 - EPS_BW <Battery/Voltage Warning>
- 1 - EPS_OFF <Subsystem Offline>

EPS_USE equals **true** when any unexpected software error has occurred. It can be true if the EPS system has detected a hardware problem it cannot correct, like firmware data corruption or a sensor value above recommended levels. This flag will tell the OBC it needs to investigate and possibly attempt a correction, or intervention.

EPS_NSE equals **true** when any event worth notifying about has occurred in the EPS subsystem. This would include events such as EPS reboots, a critical sensor value resulting in shutting of a subsystem, internal errors that have been solved, and similar notifications. This flag will tell the OBC something has happened, so it can gather event data and store it.

EPS_BW equals **true** If the voltage levels are too low, and the satellite should enter power saving mode.

EPS_OFF equals **true** if any subsystem is offline. This flag will be false only when all subsystems are online.

Transaction Details:

Command length: 1 byte
Response length: 1 byte

Command byte value: 0x1E
Response byte value: Any of the 16 hamming values.

The response value is the hamming value found by adding the flag values together. If a flag is not implemented, or false, the value for that flag is 0.

Flag	Value	Mask	Position
EPS_USE	8	(1<<3)	3
EPS_NSE	4	(1<<2)	2
EPS_BW	2	(1<<1)	1
EPS_OFF	1	(1<<0)	0

Value Example:

The EPS_USE and EPS_OFF flags are true, and the other flags are false, the hamming value to send as response would be 8+1 = Hamming 9.

Hamming 9 = 0x99.

Code Example:

```
uint_8t hamming_array[16] = {0x00, 0x1E… 0xD2, 0xE1, 0xFF};
uint_8t response_value;
uint_8t index = 0;

if(EPS_USE) index += 8;
if(EPS_NSE) index += 4;
if(EPS_BW) index += 2;
if(EPS_OFF) index += 1;

response_value = hamming_array[index];
//send response_value on I2C read request.
```

A.8.2. EPS DIAGNOSTIC

The EPS Diagnostic response consists of several values. For dynamic variables (sensors), the new value must be more recent than the last time an EPS Diagnostic request was made, or no longer than 5 minutes old.

- Package ID:
  - This is a fixed value of 0x2D, which is the same as the command byte for “EPS Diagnostic”.
- EPS ID:
  - The pre-defined ID for EPS, which is the same as its I2C slave address.
- Error flags:
  - EPS error flags.
- Event flags:
  - EPS event flags.
- EN:
  - TPS2557 enable signals.
- FAULT:
  - TPS2557 fault signals.
- Systems Online:
  - A byte with online or offline status flags for each of the primary systems. This byte should be converted to a hamming value, as with the EPS STATUS byte.
- Langmuir probe release detection:
  - Flag is set high when probe is released.
- # of restarts:
  - Number of EPS μC soft restarts.
- Sensor Values:
  - Actual sensor values, LSB first

Byte #	Name	Value
1	Package ID	0x2D
2	EPS ID	0x0E
3	System Event flags	Hamming
4	Restart flag	Hamming
5	Systems Online	Hamming
6	EN_1-4	Hamming
7	EN_5-8	Hamming
8	FAULT_1-4	Hamming
9	EN_9, 10, TCS_EN	Hamming
10	I2C Result	Actual
11	I2C Bus State	Actual
12	Sensor fault 1	Flags
13	Sensor fault 2	Flags
14-134	Sensor values	Actual
135-161	Misc. values	Arbitrary

Transaction Details:

Command length: 1 byte

Response length: 161 bytes

Command byte value: 0x2D

Response byte values: 161 bytes as described.

Code Example:

```
uint_8t response_array[48] = {0x2D, 0x0E, 0x00…  0x00, 0x00};
// copy values into array
// send response_array[I2C_Bytes_Sent] on I2C read request.
```

A.8.3. EPS APPLICATION CRC

This command tells the EPS system to return the application CRC values. If there are not any CRC values available, the EPS system will return bytes of 0x00 only. Once the transaction is completed, a new CRC value must be generated. If the new CRC is incorrect, the EPS_USE flag should be set.

Transaction Details:

Command length: 1 byte
Response length: 3 bytes

Command byte value: 0x4B
Response byte value: 24 bit CRC

Value Example:

No CRC values have been generated yet.

Byte #	1	2	3
Value	0x00	0x00	0x00

Code Example:

```
uint_8t response_array[3] = {0x00, 0x00, 0x00};
// copy CRC values into array
// send response_array[I2C_Bytes_Sent] on I2C read request.
```

A.8.4. EPS RESTART

The EPS system will restart immediately upon receiving this command.

Transaction Details:

Command length: 1 byte
Response length: 0 bytes

Command byte value: 0x55
Response byte values: Not Available

Value Example:

Not Available. No values to return.

Code Example:

```
if(0x55) //if reset command
{
  ccp_write_io( &RST.CTRL, RST_SWRST_bm );
  delay_us(1000); //prevent code to execute before reset
}

… //external assembly code:

  .global ccp_write_io
  .section .text.ccp_write_io, "ax", @progbits
  .type ccp_write_io, @function
  ccp_write_io :

  out     RAMPZ, r1              // Reset bits 23:16 of Z
  movw    r30, r24               // Load addr into Z
  ldi     r18, CCP_IOREG         // Load magic CCP value
  out     CCP, r18               // Start CCP handshake
  st      Z, r22                 // Write value to I/O register
  ret                            // Return to caller

  .size   ccp_write_io, . - ccp_write_io
```

A.8.5. EPS GROUND STATION COMMAND

The Union Ground Station will send command data to the satellite. These commands will be included in this protocol when available.

Transaction Details:

Command length: 3-257 bytes
Response length: 0-256 bytes

Command byte value: 0x66
Response byte values: Up to 256 bytes (Optional)

Action	POWER ON	POWER OFF	TOGGLE
Value	0xFF	0x00	0x0F

A.8.6. EPS POWER SUBSYSTEM

This command will tell the EPS to power off, toggle power or turn on a subsystem.

Transaction Details:

Command length: 3 bytes
Response length: 1 byte

Command byte value: 0x87
Response byte values: Systems Online flag byte.

The first byte is the command byte 0x87, telling the EPS this is a POWER SUBSYSTEM command.

Second byte tells the EPS if it should turn on, toggle, or turn off the systems.

Third byte is a flag identical in structure to the Systems Online flag that tells the EPS for which of the systems the action should be performed.

Value Example:
These 3 bytes are sent to the EPS:

Byte 1	Byte 2	Byte 3
0x87	0x0F	0x1E

This means the OBC should be toggled, meaning power off then on again. Any other action for the OBC is not valid, and the EPS should make sure the OBC is turned on if such a command is received.

Value Example 2:
These 3 bytes are sent to the EPS:

Byte 1	Byte 2	Byte 3
0x87	0x00	0xAA

This means the ADC and Payload should be powered off. Power for the OBC and COMM should not be changed.

If the OBC and COMM are both powered ON, the response value should be 1+4 = Hamming 5.

Hamming 5 = 0x55.

Flag	Value	Mask	Position
ADC	8	(1<<3)	3
COMM	4	(1<<2)	2
PLD	2	(1<<1)	1
OBC	1	(1<<0)	0

A.8.7. EPS BW settings

This command changes the Battery Warning settings. The enable bytes set the voltage (in mV) when the battery warning should be issued. The disable byte set the voltage where the Battery Warning should be cleared.

Transaction Details:

Command length: 5 byte
Response length: 0 bytes

Command byte value: 0x99
Response byte values: Not Available

The first byte is the command byte, 0x66.

Byte	Description
2	BW-set LSB
3	BW-set MSB
4	BW-clear LSB
5	BW-clear MSB

A.8.8. EPS TCS settings

This command changes the Thermal control system settings. The TCS on/off determines whether the TCS should be active or not. The TCS-enable byte set the enable threshold, in degrees Celsius. The TCS-disable set the disable threshold.

Transaction Details:

Command length: 4 byte
Response length: 0 bytes

Command byte value: 0xAA
Response byte values: Not Available

The first byte is the command byte, 0x78.

Byte	Description
2	TCS on/off (1/0)
3	TCS-enable
4	TCS-disable

A.8.9. EPS UPTIME

EPS UPTIME is a request for the EPS uptime value, which will be used during the satellite startup procedure. This will be used to confirm the required time has passed before normal operation can begin.

Transaction Details:

Command length: 1 byte
Response length: 3 bytes

Command byte value: 0xD2
Response byte values: Tick Uptime

Value Example:

The response consists of a 16 bit value that represents uptime in ticks of 10 seconds, and an 8 bit week counter.

Byte 1	Byte 2	Byte 3
0x00	0x81	0x02

If the EPS has been online for 122 250 seconds, the uptime would be 2 weeks + 1290 seconds. The values to return would then be 0x0081 and 0x02, because 0x0081 is 129 ticks of 10 seconds, and 0x02 represents 2 weeks.

A.8.10. EPS m-NLP release

Enable power to the release mechanism of the Langmuir probes.

Transaction details:

Command length: 2 bytes
Response length: 1 byte

Command byte value: 0xE1
Response byte values: Not Available

The first byte is the command byte.

The second byte tells which of the probe release mechanisms that should be enabled.

Flag	Value	Mask	Position
Probe 4	8	(1<<3)	3
Probe 3	4	(1<<2)	2
Probe 2	2	(1<<1)	1
Probe 1	1	(1<<0)	0

If the second byte is 0x00, only send the response without enabling any signals.

The response is the Langmuir-probe release-status byte.

A.9. OBC – ON-BOARD CONTROLLER & DATA HANDLING

Command Name	Command	Response
OBC Application CRC	0x4B	None
OBC Restart	0x55	None
OBC Memory Access	0x66 + 256 bytes	256 bytes
OBC Transfer Complete	0x87	None
OBC Transfer Diagnostic	0x99 + 2 bytes	256 bytes
OBC Transfer Event Log	0xAA + 2 bytes	256 bytes
OBC Transfer Payload	0xB4 + 1 bytes	None
OBC Update Time	0xD2 + 5 bytes	None

A.9.1. OBC APPLICATION CRC

This command tells the OBC to generate a new Application CRC value, and check that it is a match.

Transaction Details:

Command length:	1 byte
Response length:	0 bytes
Command byte value:	0x4B
Response byte value:	None

A.9.2. OBC RESTART

The OBC system will restart immediately upon receiving this command.

Transaction Details:

Command length: 1 byte
Response length: 0 bytes

Command byte value: 0x55
Response byte values: Not Available

Value Example:

Not Available. No values to return.

Code Example:

```
if(0x55) //if reset command
{
  ccp_write_io( &RST.CTRL, RST_SWRST_bm );
  delay_us(1000); //prevent code to execute before reset
}

… //external assembly code:

  .global ccp_write_io
  .section .text.ccp_write_io, "ax", @progbits
  .type ccp_write_io, @function
  ccp_write_io :

  out     RAMPZ, r1              // Reset bits 23:16 of Z
  movw    r30, r24               // Load addr into Z
  ldi     r18, CCP_IOREG         // Load magic CCP value
  out     CCP, r18               // Start CCP handshake
  st      Z, r22                 // Write value to I/O register
  ret                            // Return to caller

  .size   ccp_write_io, . - ccp_write_io
```

A.9.3. OBC MEMORY ACCESS

The UiO Ground Station will access the OBC memory directly, for either a read or a write operation.

Transaction Details:

Command length: 3-257 bytes
Response length: 0-256 bytes

Command byte value: 0x66
Response byte values: Up to 256 bytes (Optional)

A.9.4. OBC TRANSFER COMPLETE

This command is an ACK from the COMM system to start transfer of the next package, if any.

Transaction Details:

Command length: 1 byte
Response length: 0 bytes

Command byte value: 0x87
Response byte values: None

A.9.5. OBC TRANSFER DIAGNOSTIC

This command will tell the OBC to send diagnostic data to the ground station.

Byte#	Type	Value
1	Command	0x99
2	Diagnostic Index	0-255
3	Number of Packages	0-255

Transaction Details:

Command length: 3 bytes
Response length: 0/256 bytes

Command byte value: 0x99
Response byte values: None OR 256 byte Diagnostic Data

Value Example:

If the diagnostic index is 53, and number of packages is 21, the OBC will begin transmitting the diagnostic packages from index 53 to and including 73 to the ground station.

A.9.6. OBC TRANSFER EVENT LOG

This command will tell the OBC to send the event logs to the ground station.

Byte#	Type	Value
1	Command	0xAA
2	Event Index	0-255
3	Number of Packages	0-255

Transaction Details:

Command length: 3 byte
Response length: 0/256 bytes

Command byte value: 0xAA
Response byte values: None OR 256 byte Event Data

Value Example:

If the event index is 3, and number of packages is 0, the OBC will not send anything to the ground station. The 256 byte response would then be the event package at index 3.

A.9.7. OBC TRANSFER PAYLOAD

This command will tell the OBC to collect and send payload data to the ground station.

Byte#	Type	Value
1	Command	0xB4
2	Number of Packages	1-256

Transaction Details:

Command length: 2 byte
Response length: 0 bytes

Command byte value: 0xB4
Response byte values: None

Value Example:

If the number of packages is 16, the OBC will get 16 payload packages from the payload system, and send them to the ground station. If the value is 0, the OBC will send 256 packages.

A.9.8. OBC UPDATE TIME

OBC UPDATE TIME will update the internal satellite time value, which will be used to timestamp data packages and events.

Transaction Details:

Command length: 4 bytes
Response length: 0 bytes

Command byte value: 0xD2
Response byte values: None

Value Example:
The COMM system sends the current satellite time to the OBC. This data is presented as a 10-second tick timestamp with 16 bits, and a week counter with 8 bits.

The OBC will not respond.

Time will be represented as a time-lapse since January 1st 2012. (To be revised later).
Example on transaction for April 10th 2012 at 00:00:

0xD2 (command byte value)
0x43
0x80 (0x4380 ticks or 172800 seconds or 48 hours since start of week 15)
0x0F (week 15)

Byte 1	Byte 2	Byte 3	Byte 4
0xD2	0x43	0x80	0x0F

A.10. PAYLOAD – LANGMUIR PROBE

Command Name	Command	Response
PAYLOAD Ground Station Command	0x66 + 256 bytes	256 bytes
PAYLOAD Get Data	0x87	256 bytes
PAYLOAD Update Time	0xD2 + 5 bytes	5 bytes

A.10.1. PAYLOAD GROUND STATION COMMAND

The UiO Ground Station will send command data to the satellite. These commands will be included in this protocol when available.

Transaction Details:

Command length:	3-257 bytes
Response length:	0-256 bytes
Command byte value:	0x66
Response byte values:	Up to 256 bytes (Optional)

A.10.2. PAYLOAD GET DATA

This command will return a payload data package, either housekeeping or scientific data.

Transaction Details:

Command length:	1 byte
Response length:	256 bytes
Command byte value:	0x87
Response byte values:	256 byte Payload Data

A.10.3. PAYOAD UPDATE TIME

PAYLOAD UPDATE TIME will update the internal payload time value, which will be used to timestamp data packages and events.

Transaction Details:

Command length: 6 bytes
Response length: 5 bytes

Command byte value: 0xD2
Response byte values: Old Time Value

Value Example:

The OBC sends the current satellite time to the Payload. This data is presented as a 1/1024 second high-resolution timestamp with 16 bits, a 10-second tick timestamp with 16 bits, and a week counter with 8 bits.

The Payload will respond with the previous time value.

Time will be represented as a time-lapse since January 1st 2012. (To be revised later).
Example on transaction for April 10th 2012 at 00:00:

0xD2 (command byte value)
0x00
0x00 (0x0000 1/1024 seconds between ticks)
0x43
0x80 (0x4380 ticks or 172800 seconds or 48 hours since start of week 15)
0x0F (week 15)

Byte 1	Byte 2	Byte 3	Byte 4
0xD2	0x43	0x80	0x0F

A.11. DATA PACKAGE FORMATS

A.11.1. Beacon & Diagnostic

Byte #	Name	Value
1	Package ID	0x87
2	OBC ID	0x12
3	Week	8 bit value
4-5	Tick	16 bit value
6-15	OBC Data	Arbitrary
16-28	COMM Data	See COMM documentation
29-62	EPS Data 1	See EPS documentation
63-64	CRC Value 1	16 bit CRC
65-126	EPS Data 2	See EPS documentation
126-128	CRC Value 2	16 bit CRC
130-189	EPS Data 3	See EPS documentation
191-192	CRC Value 3	16 bit CRC
193-254	ADCS Data	See ADCS documentation
255-256	CRC Value 4	16 bit CRC

A.11.2. System Event

An event package sent to ground contains 16 system events with the following content:

Byte #	Name	Value
1	Package ID	0xCC
2	System ID	Any
3	Week	8 bit value
4-5	Tick	16 bit value
6-16	Event Data	Arbitrary

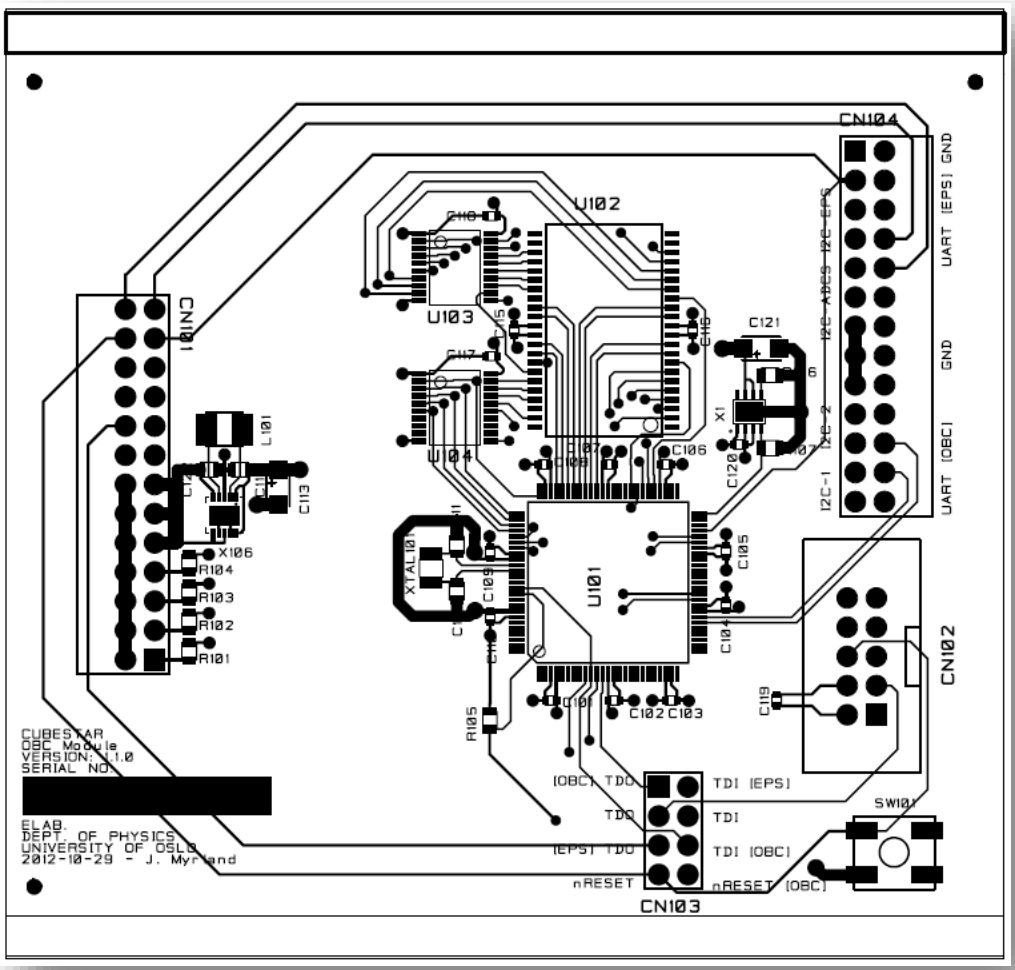
A.11.3. Payload

Byte #	Name	Value
1	Package ID	0xB4
2-256	Scientific Data	Any

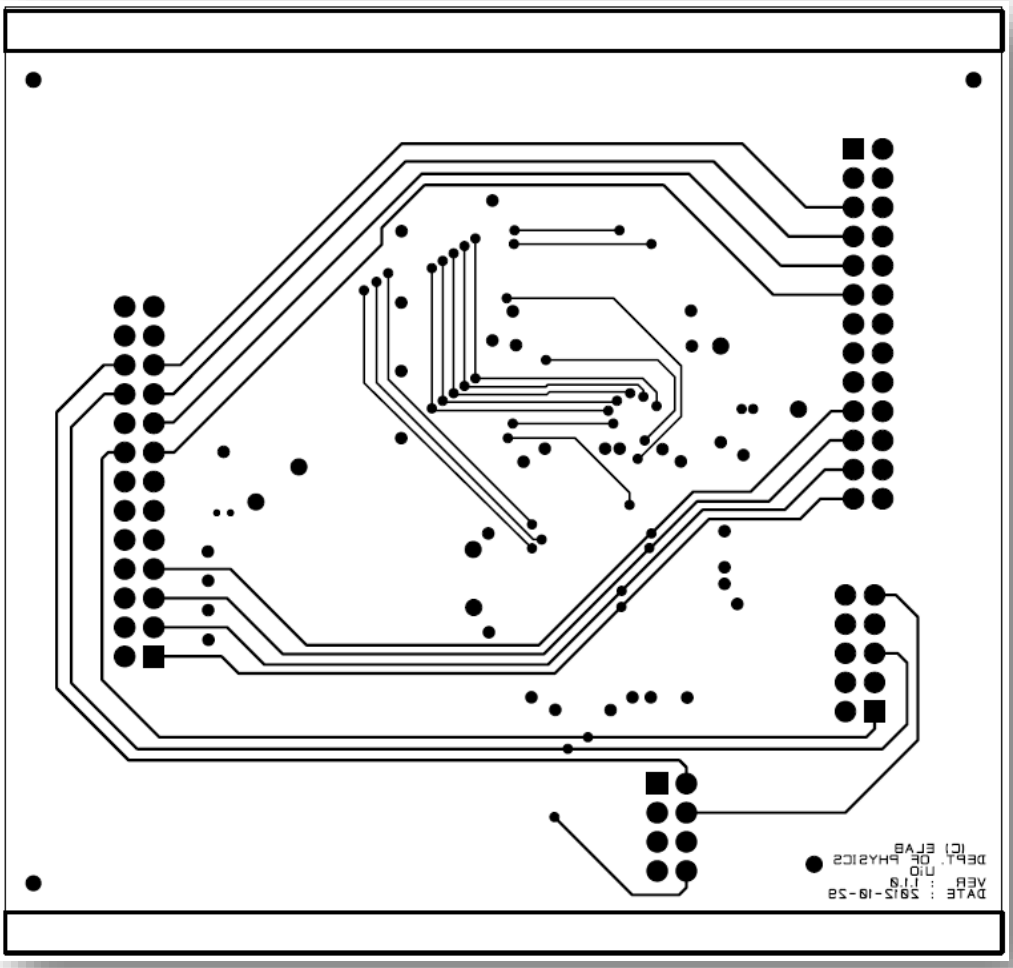
Appendix B: OBC Hardware

B.1. VERSION 1.1.0

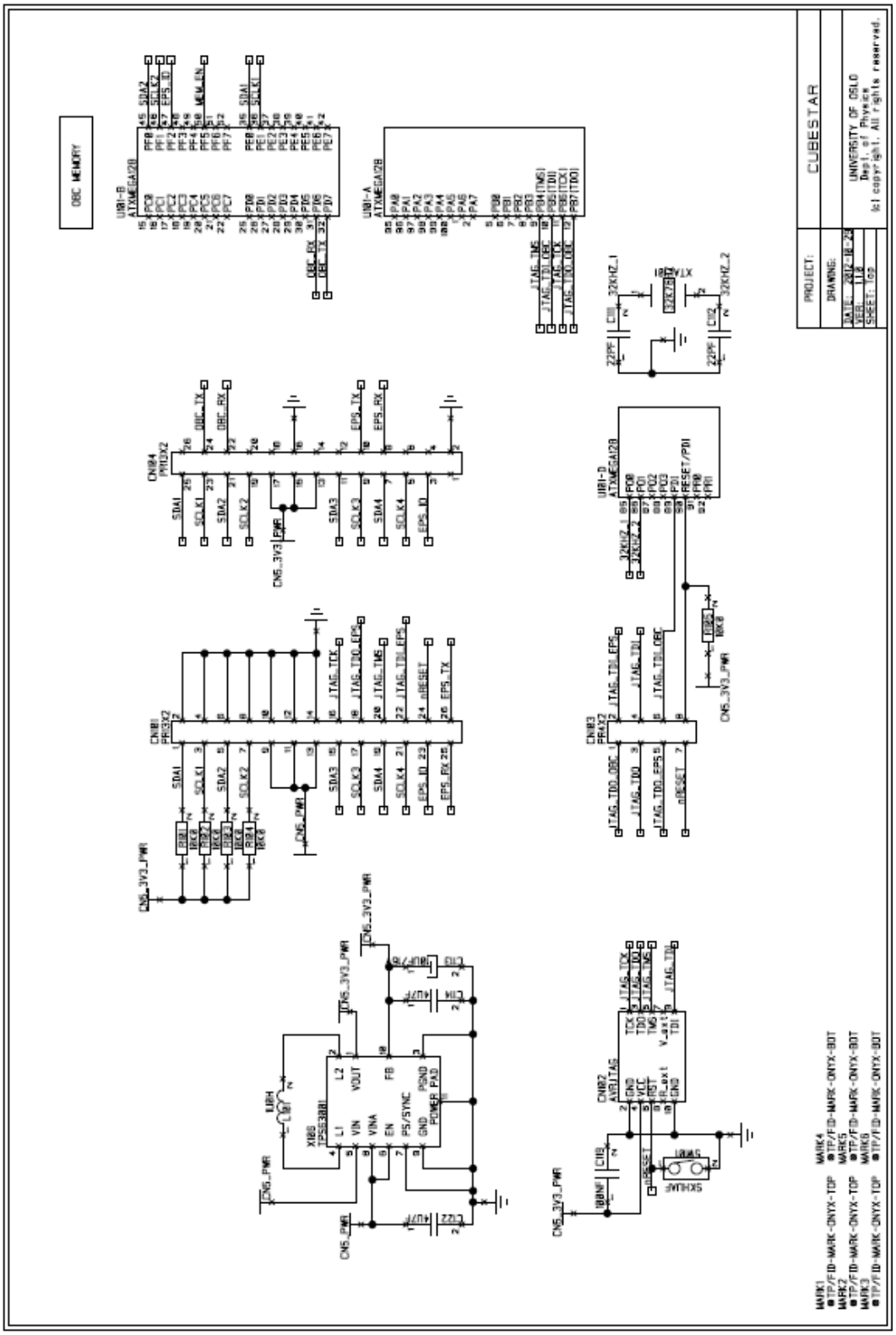
B.1.1. Layout – Top Copper and Silk



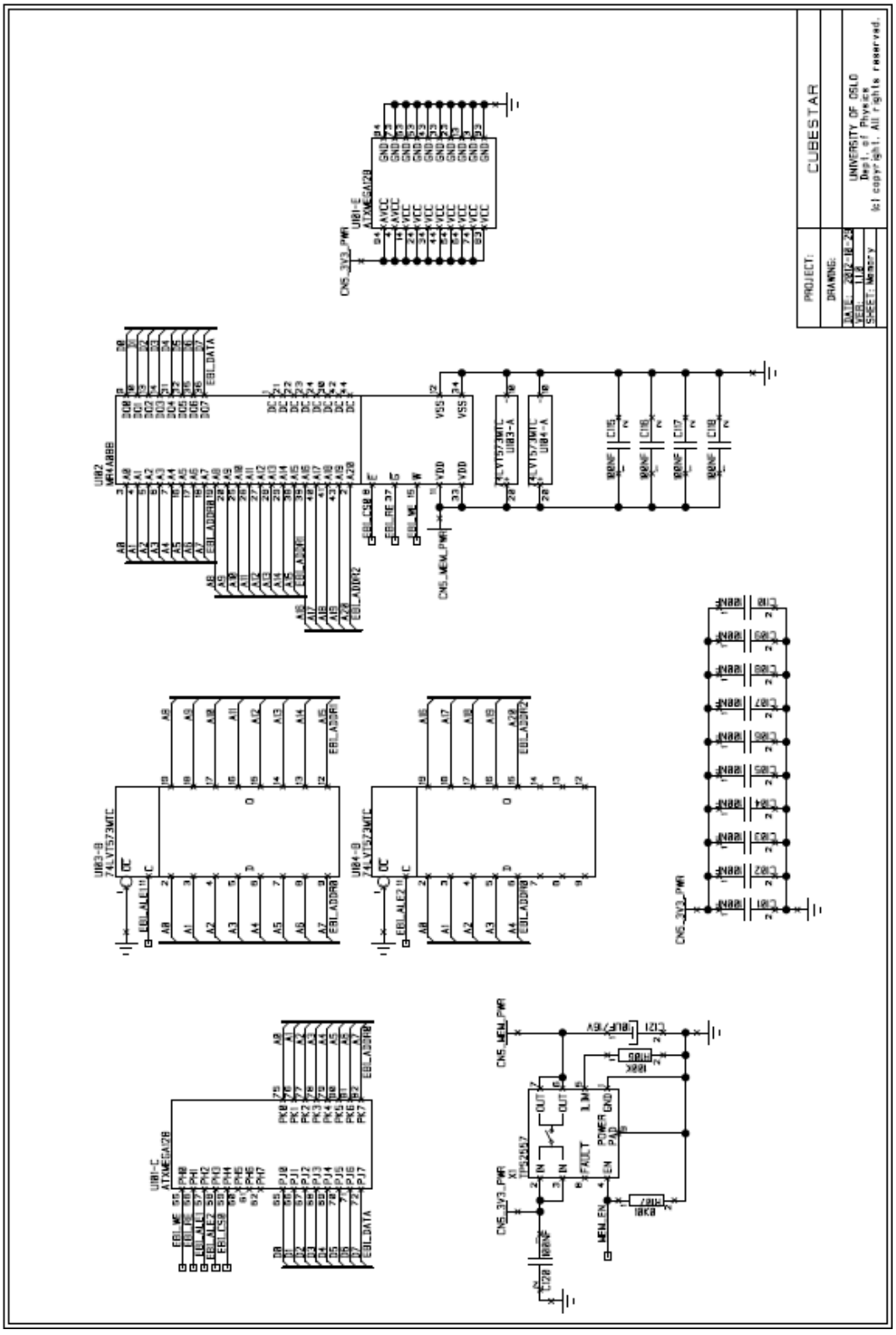
B.1.2. Layout – Bottom Copper and Silk



B.1.1. Schematic – Top

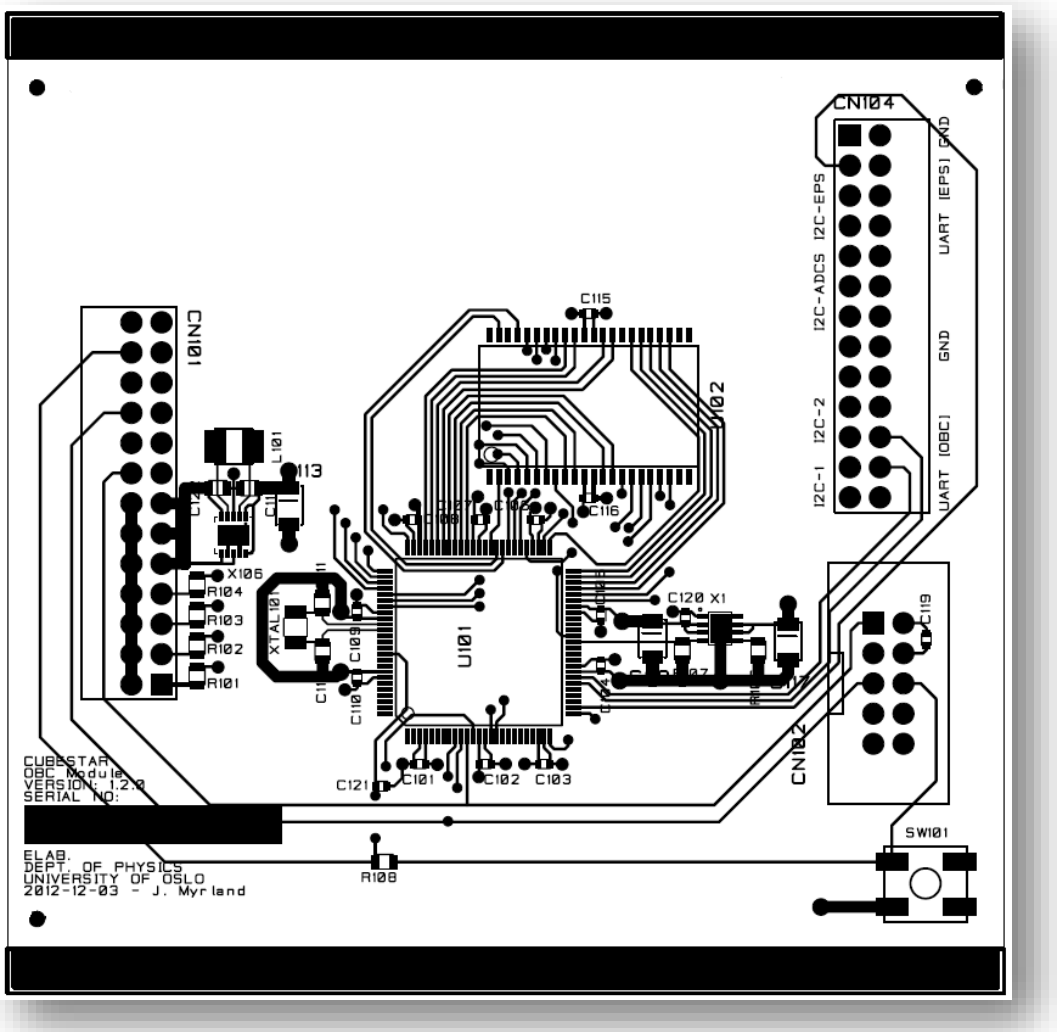


B.1.1. Schematic – Memory

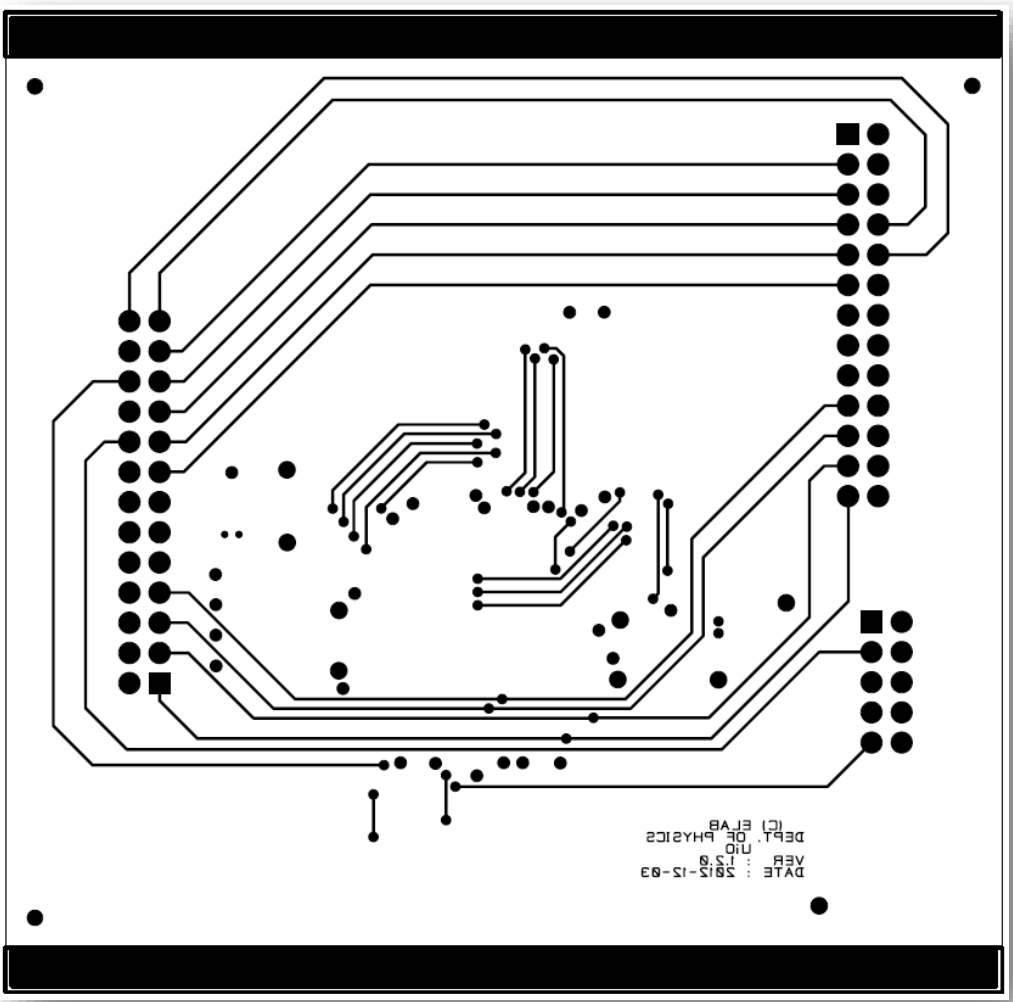


B.2. Version 1.2.0

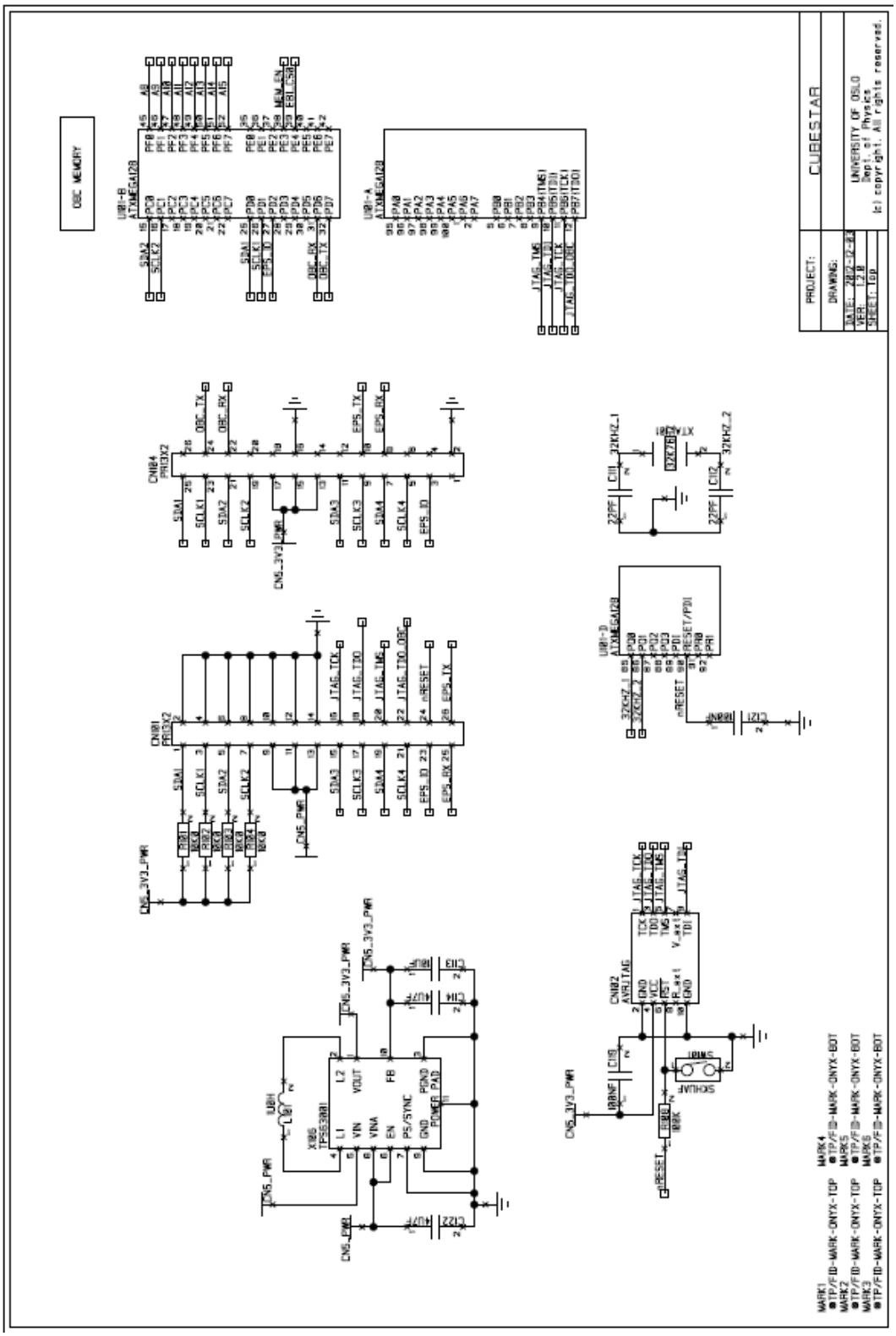
B.2.1. Layout – Top Copper and Silk



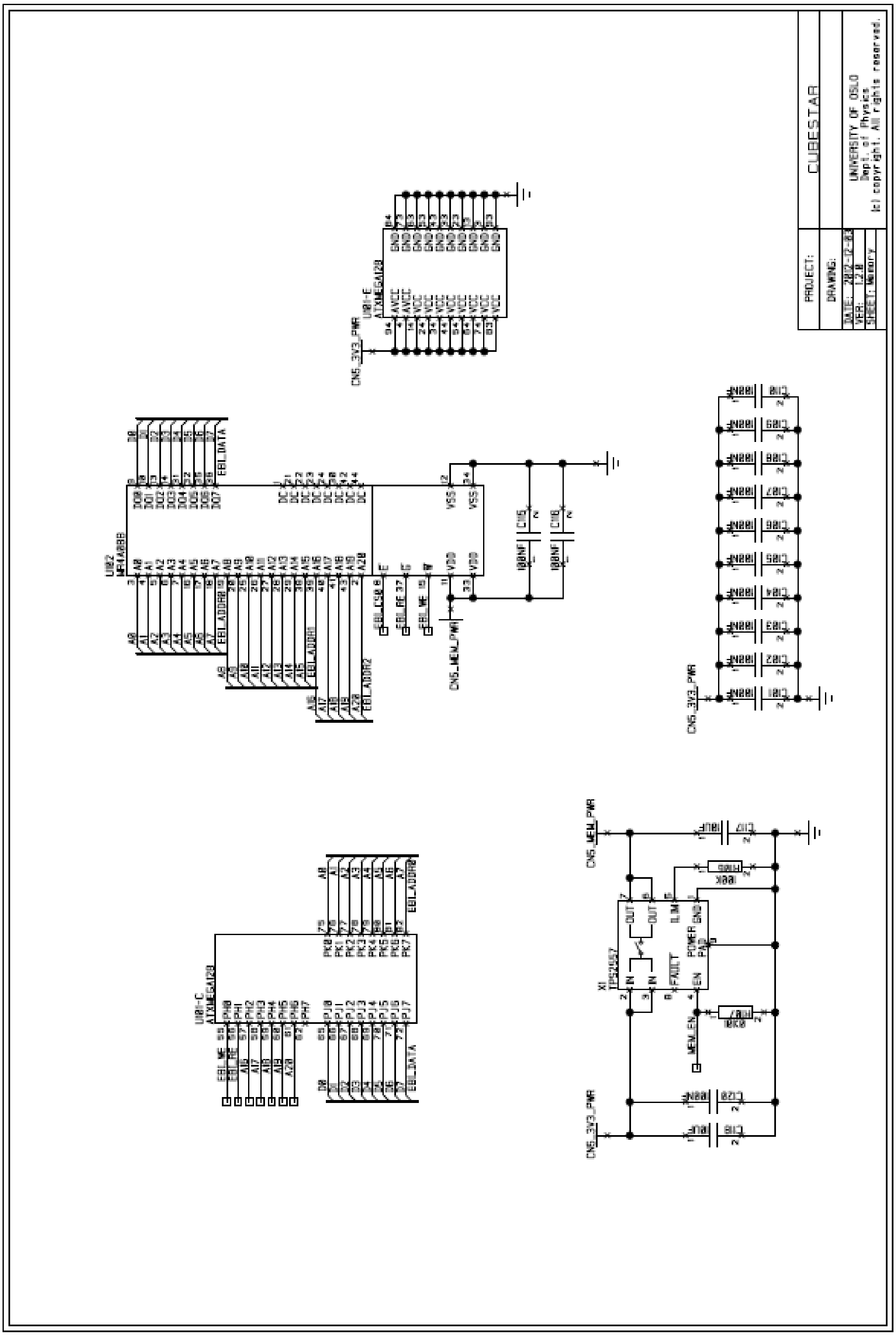
B.2.2. Layout – Bottom Copper and Silk



B.2.1. Schematic – Top



B.2.1. Schematic – Memory



B.3. BILL OF MATERIALS

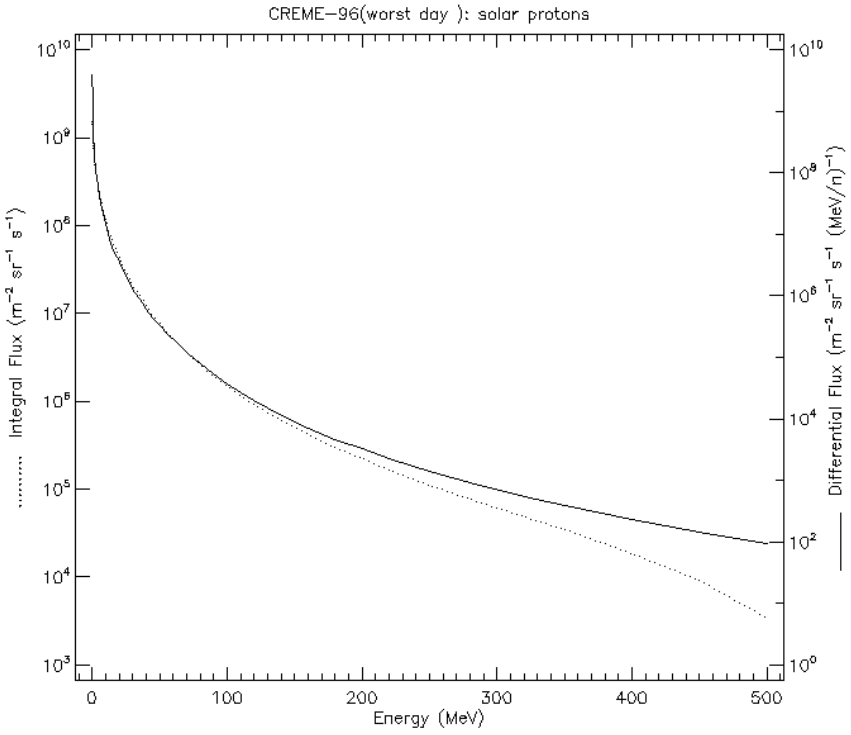
Description	Store	Order ID	Layout Names	Quantity
ATXMEGA128A1U	Digikey	ATXMEGA128A1U-AU-ND	U101	1
MRAM/MR2A08B	Digikey	819-1003-ND	U102	1
XTAL/32K76HZ/CM315	Digikey	300-8751-1-ND	XTAL101	1
POW/TPS63001	Digikey	296-19643-1-ND	X106	1
POW/TPS2557	Farnell	F-1782842	X1	1
IND/1U5H/LQH3N	Digikey	490-5343-1-ND	L101	1
CAP/18PF/0603R	Digikey	399-9017-1-ND	C111-112	2
CAP/100NF/0402R	Digikey	399-6828-1-ND	C101-110 C115-C116, C119-121	15
CAP/4U7F/0603R	Farnell	F-1833806	C114 C122	2
CAP/10UF/1206R	Elfa	E-65-834-39	C113, C117, C118	3
RES/100K/0603R	Elfa	E-60-452-64	R106, R108	2
RES/10K0/0603R	Elfa	E-60-450-25	R101-104 R107	5
SW/SKHUAF/SMD	Elfa	E-35-790-18	SW101	1
CON/AVRJTAG		X-XX-XXX-XX	CN102	1
CON/PR13X2PIN/HORIZ	Elfa	E-43-714-31	CN101 CN104	2

Appendix C: SPENVIS

Various plots and graphs have been created using SPENVIS. The standard orbit parameters are 300 km altitude and 85 degrees inclination unless otherwise specified. Mission duration was set between 2011-01-01 and 2011-01-21.

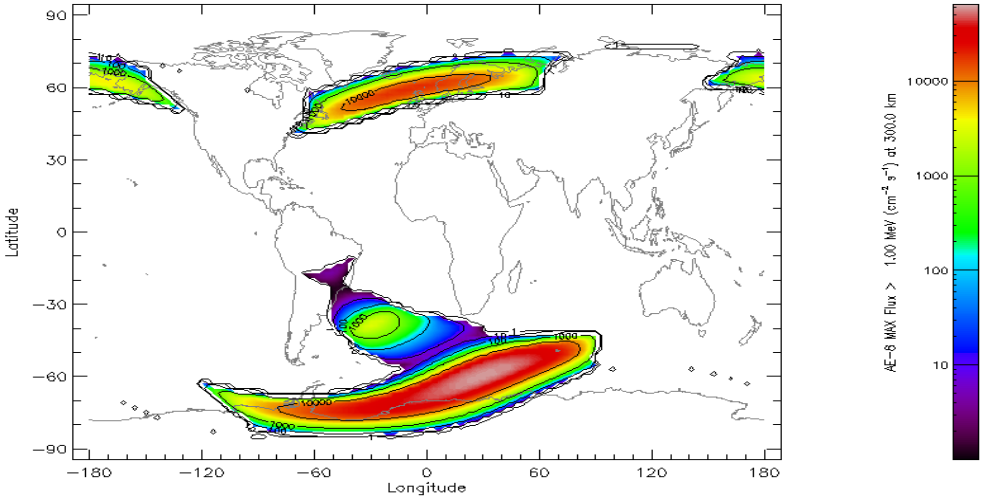
C.1. WORST DAY PROTON FLUX

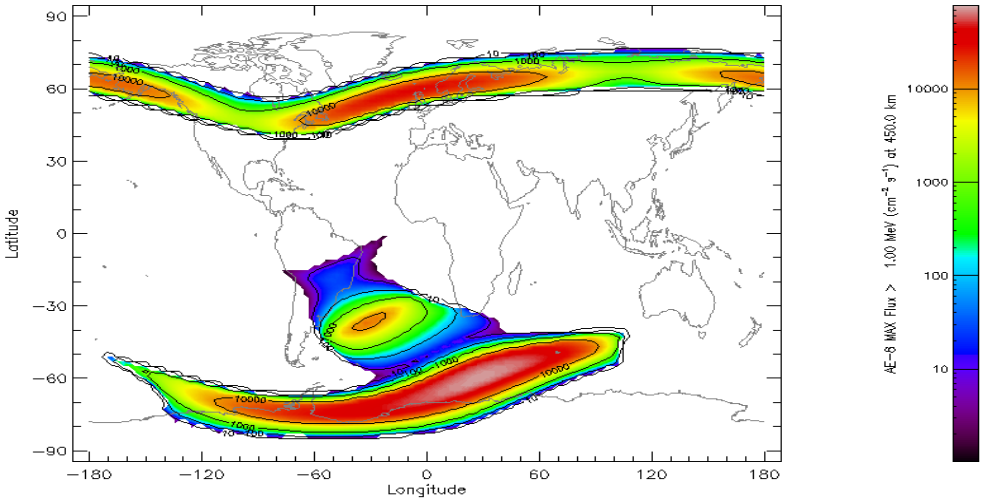
This chart shows the worst day SPE proton flux, using the CRÈME-96 model, and the standard orbit parameters.



C.2. ELECTRON FLUX

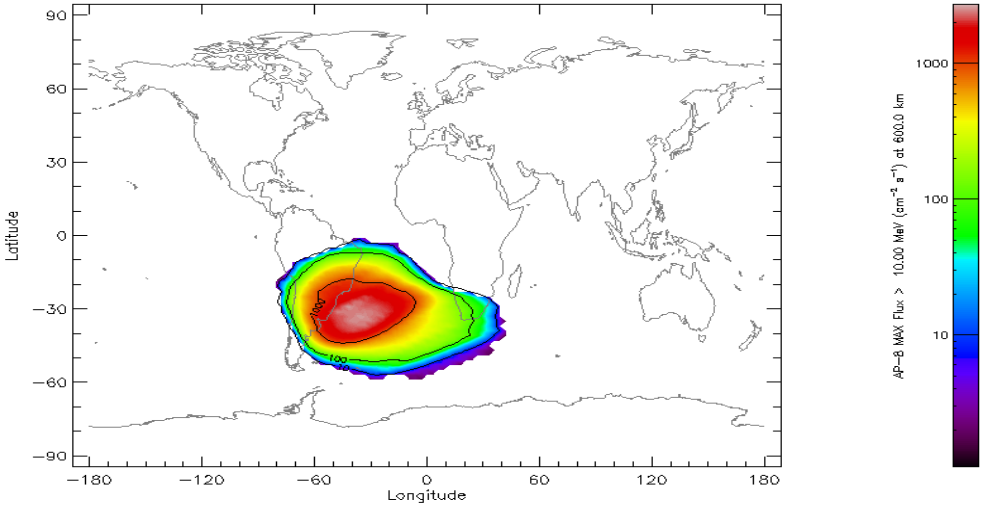
These charts show the electron flux at 300 km and 450 km orbit. Only electrons with >1 MeV are included. The analysis is done using the AE-8 Solar Maximum radiation belt model.

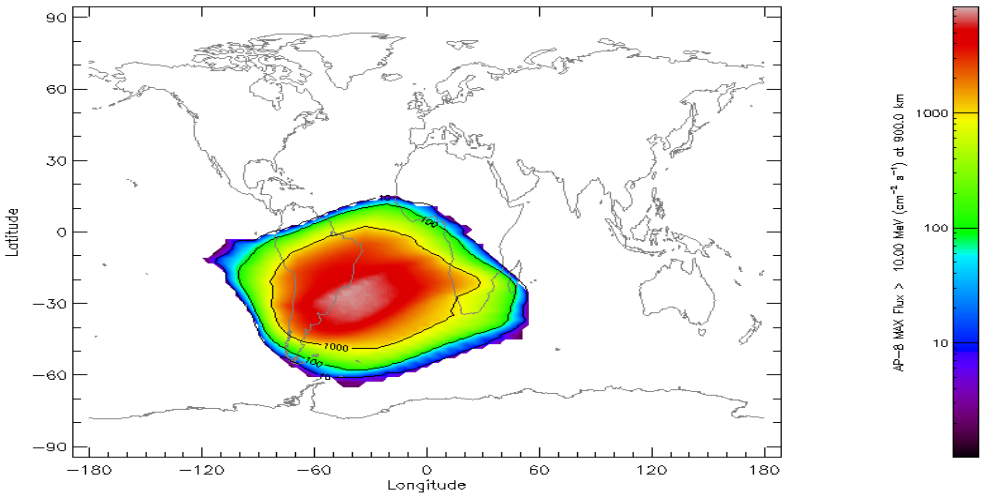




C.3. PROTON FLUX

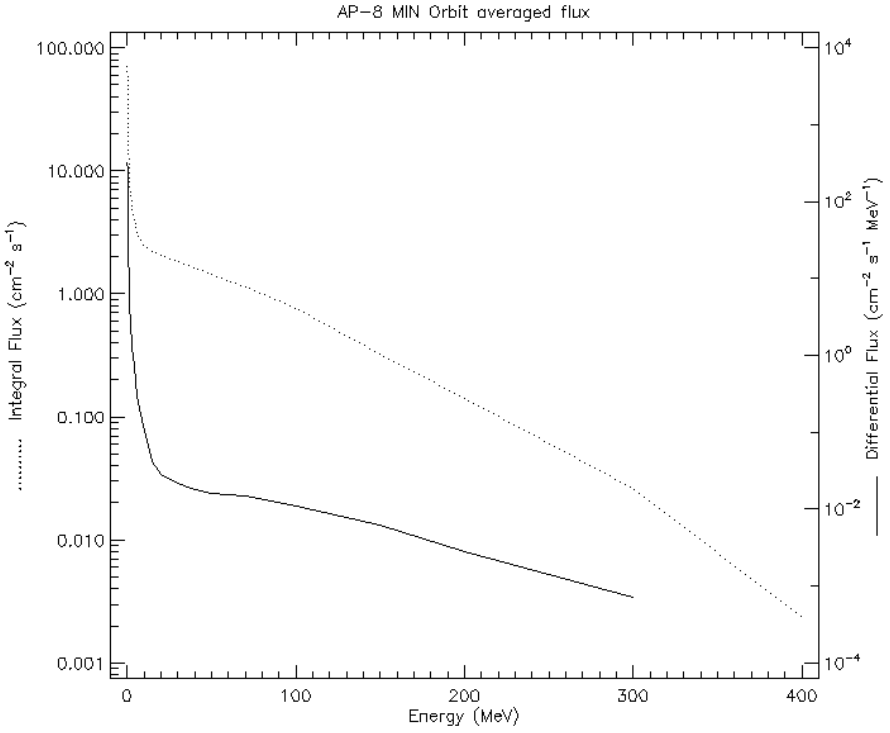
These charts shows the proton flux at 600 km and 900 km orbit. Only protons with >10 MeV are included. The analysis is done using the AP-8 Solar Maximum radiation belt model. The proton flux and area increases significantly at higher altitudes, at 900 km the flux is ~40 times the level found at 300 km.

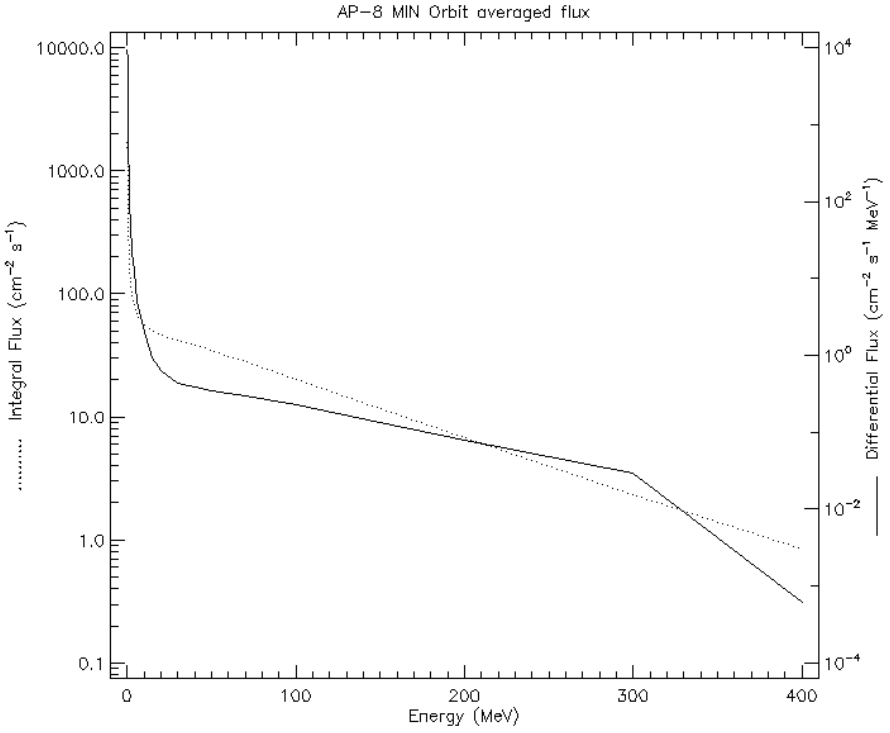




C.4. AVERAGE PROTON FLUX

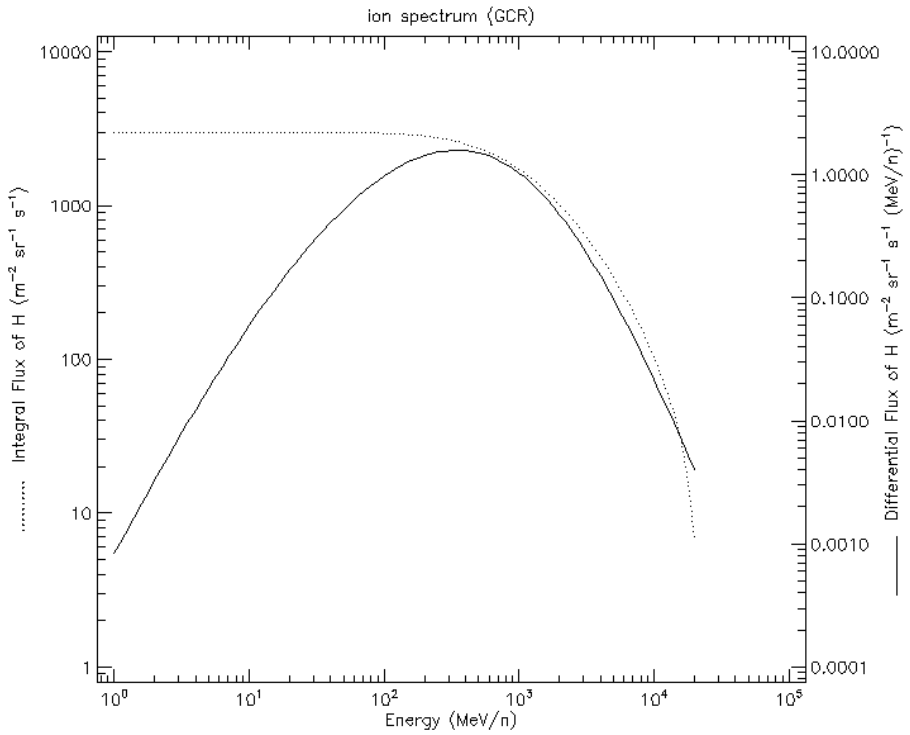
These charts show the average proton flux for the standard and 600 km orbit.

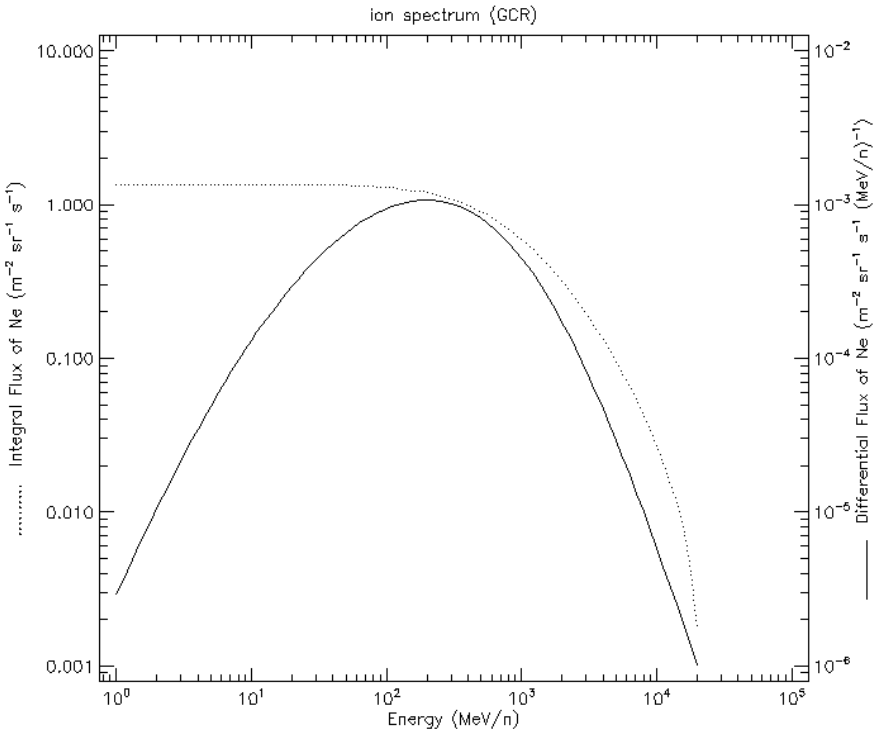


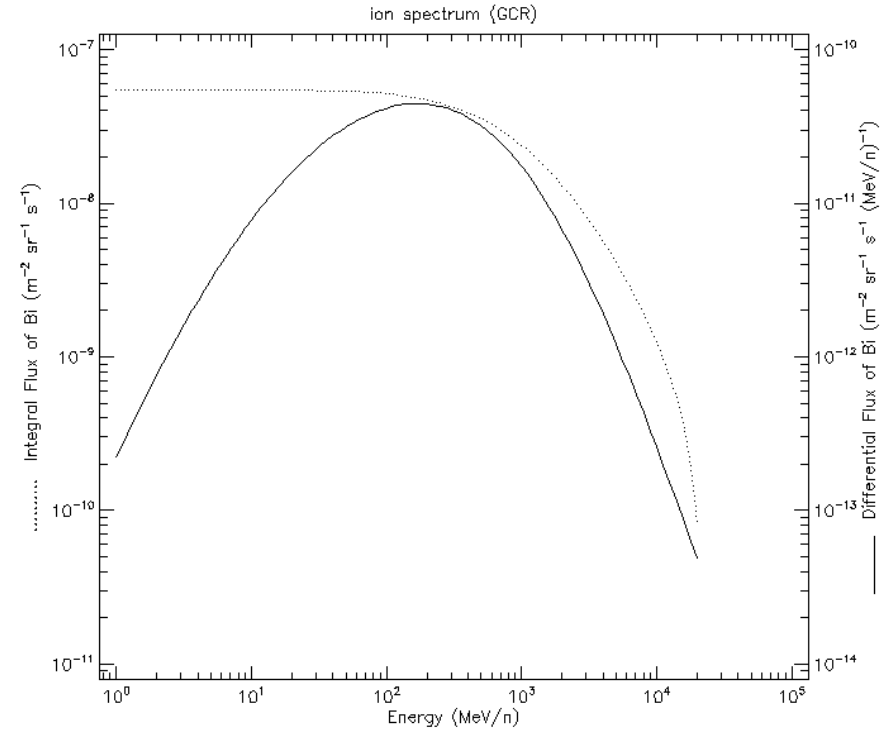


C.5. GALACTIC COSMIC RAYS

These charts show the ion spectrum for Helium (Z=1), Neon (Z=10) and Bismuth (Z=83). These are found using the standard orbit parameters. Bismuth was the Ion with the highest Z value that had any values on the Ion spectrum.




ion spectrum (GCR)
Integral Flux of Ne (m^{-2} sr^{-1} s^{-1})
Differential Flux of Ne (m^{-2} sr^{-1} s^{-1} (MeV/n)$^{-1}$)
Energy (MeV/n)
10.000
1.000
0.100
0.010
0.001
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}


ion spectrum (GCR)
Integral Flux of Bi (m^{-2} sr^{-1} s^{-1})
Differential Flux of Bi (m^{-2} sr^{-1} s^{-1} (MeV/n)$^{-1}$)
Energy (MeV/n)
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}